Grado en Matemática Computacional

Estancia en Prácticas y Proyecto Final de Grado

Minería de datos aplicada a Twitter y análisis de sentimientos mediante algoritmos de inteligencia artificial

Autor:
Luis Piñón Ferrer

Supervisor:
Ricardo Chalmeta Rosaleñ
Tutor académico:
María Victoria Ibáñez Gual

Fecha de lectura: Septiembre de 2018
Curso académico 2017/2018

Resumen

En este documento se explican los fundamentos teóricos de las principales técnicas de minería de datos e inteligencia artificial aplicados al análisis de texto procedente de redes sociales. Se introducen los principales algoritmos de Análisis de Asociaciones, Clustering (Agrupamiento) y Clasificación y Predicción. También se realizan dos aplicaciones prácticas: una para el sondeo de redes sociales para analizar los sentimientos de los usuarios respecto a cierto tema y otra para la búsqueda de potenciales clientes que muestran interés o buscan activamente soluciones a sus problemas.

Palabras clave

Mineria de datos, minería de sentimientos, minería de texto, inteligencia artificial.

Keywords

Data mining, sentiment mining, text mining, artificial intelligence.

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El uso de las nuevas tecnologías, y con ello el uso de las redes sociales, ha crecido exponencialmente en los últimos años. Servicios gratuitos como Facebook o Twitter se han convertido en la principal herramienta de la población para expresar lo que piensan, lo que hacen o simplemente conocer lo que hacen los otros usuarios. Estos servicios, sin embargo, ganan grandes cantidades de dinero vendiendo los datos de sus usuarios a empresas publicitarias que se anuncian en las plataformas, segmentando sus anuncios a audiencias muy concretas, consiguiendo así ratios de conversión (interés de los usuarios) muy elevados.

Dichas compañías utilizan algoritmos de minería de datos e inteligencia artificial para analizar las publicaciones (imágenes, texto, videos, etc...) de los usuarios y las interacciones con otras publicaciones para conocer los gustos y sus aficiones. Con toda esta información, los algoritmos que controlan las redes sociales se encargan de mostrar las publicaciones que más pueden interesar a cada usuario y conseguir así su atención para continuar utilizando el servicio. Además, aprovechan para venderlos a los anunciantes que desean hacer llegar su producto a un público muy concreto (interesados en un tema especifico, según su género o edad, etc...).

En el desarrollo de este proyecto realizaremos una aproximación a dichas técnicas de minería de datos aplicadas a publicaciones en redes sociales (minería de texto o minería de opinión). Primero una aproximación teórica y, a continuación, dos aproximaciones prácticas.

Por una parte, se utiliza para encontrar potenciales clientes, interesados en cierto tema o que están buscando activamente una solución a su problema. Realizamos un análisis de las publicaciones relacionadas con los productos (soluciones) ofrecidas por la empresa en la que se

ha realizado la estancia en prácticas.

Por otra parte, se desarrolla un sistema capaz de analizar las publicaciones, relacionadas con una temática concreta, y clasificarlas en positivas y negativas para así obtener una puntuación de satisfacción del usuario.

La principal motivación del proyecto ha sido aprender y desarrollar una sistema que aplique técnicas de minería de datos a texto procedente de publicaciones de redes sociales para la explotación de una información todavía poco conocida. La minería de opinión aplicada a redes sociales permite al organizador de un evento o personaje público conocer el grado se satisfacción del cliente (población) sin tener que realizar una encuesta intrusiva en el ciudadano.

Capítulo 2

Estancia en prácticas

2.1. Introducción

La estancia en prácticas se ha realizado en la Fundación Universitat Jaume I - Empresa que quiere publicar los servicios que ofrece (soluciones) desde los equipos de investigación de la propia universidad.

El principal objetivo es el de crear un portal de búsqueda de soluciones para que las empresas puedan encontrar soluciones a sus demandas. Dicho portal permite:

- Buscar soluciones y patentes de la universidad y sus equipos de investigación. La búsqueda se puede realizar mediante *keywords* (palabras clave) que se introducen en el buscador, por áreas de la organización sobre las que se aplican, sobre las áreas de conocimiento y mercados potenciales.

- Introducir los datos por parte de los investigadores. Un panel de administración para publicar las soluciones que ofrece la organización.

Además, se desarrolla un bot rastreador de redes sociales capaz de detectar potenciales clientes interesados en las soluciones ofrecidas por la universidad. El bot navega por las redes sociales como si se tratara de un usuario normal y va analizando las publicaciones de los usuarios detectando usuarios potenciales y notificando a los grupos de investigación para ponerse en contacto.

2.2. Objetivos del proyecto formativo

Se realiza el desarrollo de la app web utilizando un patrón de programación MVC (Modelo, Vista, Controlador) con PHP, AngularJS, MySQL, HTML, CSS y JavaScript. Para el bot se utiliza phantomjs para la extracción de los datos y KNIME y RStudio para el procesado y visualización de los datos.

2.3. Explicación detallada del proyecto realizado en la empresa

En esta sección se desarrolla el proyecto realizado en la empresa de prácticas:

2.3.1. Metodología y definición de tareas

Programación de las vistas: maquetación de la disposición de los elementos (menús, textos, formularios, etc...) en las diferentes ventanas de la aplicación utilizando HTML5 y definición de los estilos visuales (colores, fuentes, tamaños, etc ...) con CSS.

Modelado Lógico, Conceptual y Físico de la base de datos: modelar e implementar una base de datos relacional transaccional utilizando la tecnología MySQL para almacenar la información de las soluciones ofrecidas por los investigadores. Se realiza a partir de la información que se puede obtener y se debe almacenar de las vistas del panel de administración y según las formas de búsqueda solicitadas.

Programación de la lógica de la web: configuración de las rutas, vistas, peticiones y tratamiento de los datos utilizando PHP montado en un servidor Apache.

Conexión de los datos de las bases de datos con las vistas: se conectan las respuestas obtenidas de las bases de datos con las vistas utilizando la tecnología AngularJS que permite mostrar la información de forma dinámica, sin necesidad de recarga de la página, utilizando JavaScript desde el navegador.

Programación de un bot rastreador: desarrollo de un bot utilizando phantomjs y JavaScript que simule la interacción de un usuario real con las redes sociales y obtenga y almacene los datos de las publicaciones.

Minado (procesado) de los datos: aplicación de técnicas de mineria de datos para procesar las publicaciones de los usuarios y detectar clientes potenciales.

2.3.2. Planificación temporal de las tareas

- **1ª Quincena**: entrevista con el cliente (Fundación Universitat Jaume I) para elicitación de requisitos y definición de vistas. Programación y validación de las vistas.
- **2ª Quincena**: modelado de la base de datos siguiendo los requisitos del cliente y acoplándose a los inputs (campos de entrada) de los formularios del panel de administración. También teniendo en cuenta los criterios de búsqueda por diferentes taxonomías (categorías).
- **3ª - 4ª Quincena**: programación de la lógica web. Rutas, control de acceso con usuarios a través del portal de la Universidad, sistema de indexación y búsqueda, creación de peticiones y respuestas para obtener y procesar datos de forma dinámica.
- **5ª Quincena**: conexión de las vistas con la lógica web. Comprobar que la información almacenada en las bases de datos se muestra correctamente en la aplicación web y que el sistema de búsqueda y creación de soluciones dinámico (con AngularJS) funciona correctamente.
- **6ª Quincena**: validación de la wep app con el cliente y realización de cambios y correcciones.
- **7ª Quincena**: desarrollo de bot rastreador de redes sociales ajustado para Twitter y Facebook y almacenado de los datos para su posterior procesado.
- **8ª - 9ª Quincena**: investigación sobre técnicas de minería de datos e inteligencia artificial. Programación del sistema de minado (análisis) de publicaciones y testeo con los datos obtenidos con el bot rastreador.
- **10ª Quincena**: ajuste del sistema de rastreo y minado de información y automatización del sistema.

2.3.3. Estimación de recursos del proyecto

Se utiliza un Intel Nuc i5 como ordenador de trabajo en la empresa; para la programación y procesado de los datos. El entorno LAMP (Lynux el sistema operativo, Apache el servidor web, MySQL el gestor de base de datos y PHP el lenguaje de programación) para el testeo de la aplicación en un entorno local.

Para el entorno remoto (distribución) se ha contratado un servidor por 1 euro al mes en Aruba (empresa de cloud VPS, servidores virtuales con máquinas preconfiguradas y discos SSD) que nos ofrece 1 vCPU, 1 GB de memoria RAM, 20 GB de almacenamiento SSD y 2 TB de tráfico mensual. Más que suficiente para probar y lanzar la web app.

2.3.4. Grado de consecución de los objetivos propuestos

La primera de las etapas ha consistido en diseñar vistas (ventanas) que satisfacen los requisitos del cliente y permiten búscar, consultar e introducir información de una forma fácil y eficiente.

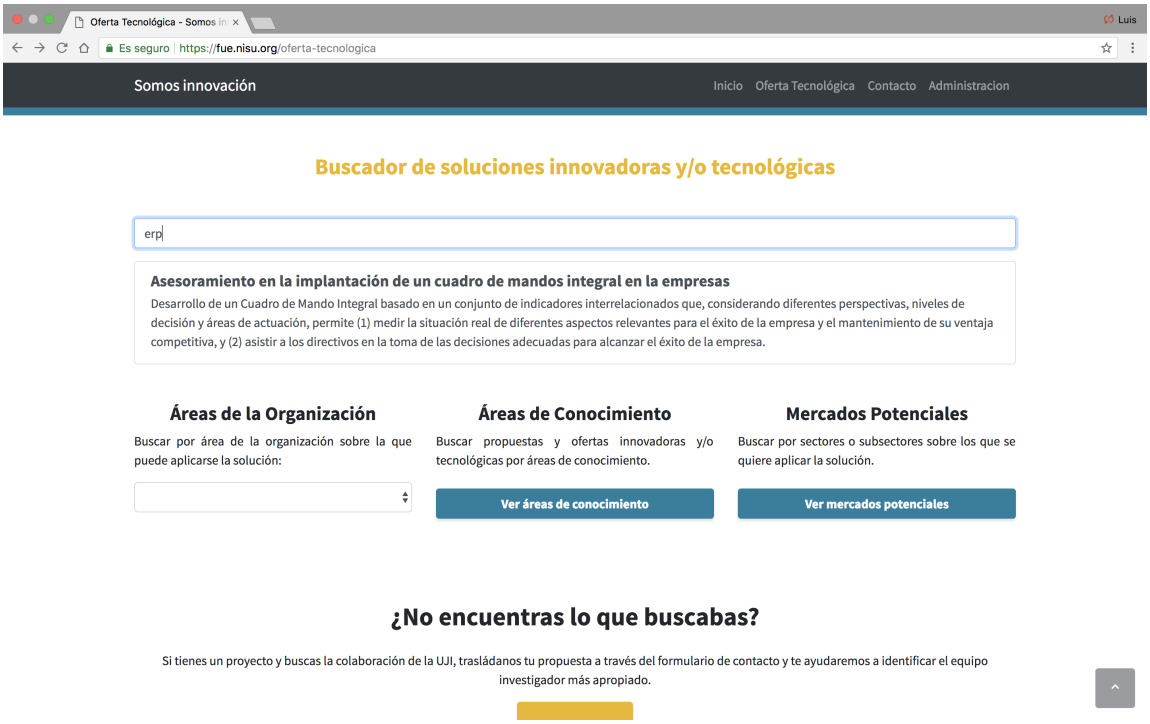

Figura 2.1: Buscador dinámico y por taxonomías (categorías).

El buscador (Figura 2.1) permite realizar búsquedas por las diferentes taxonomías (categorías) solicitadas y utilizando palabras claves en un buscador interactivo que nos ofrece respuestas instantáneas mostrando resultados de búsqueda a medida que el usuario introduce caracteres en el buscador.

Se ha diseñado una base de datos (Figura 2.2) capaz de almacenar la información de las propuestas y las relaciones con las categorías y términos de indexación.

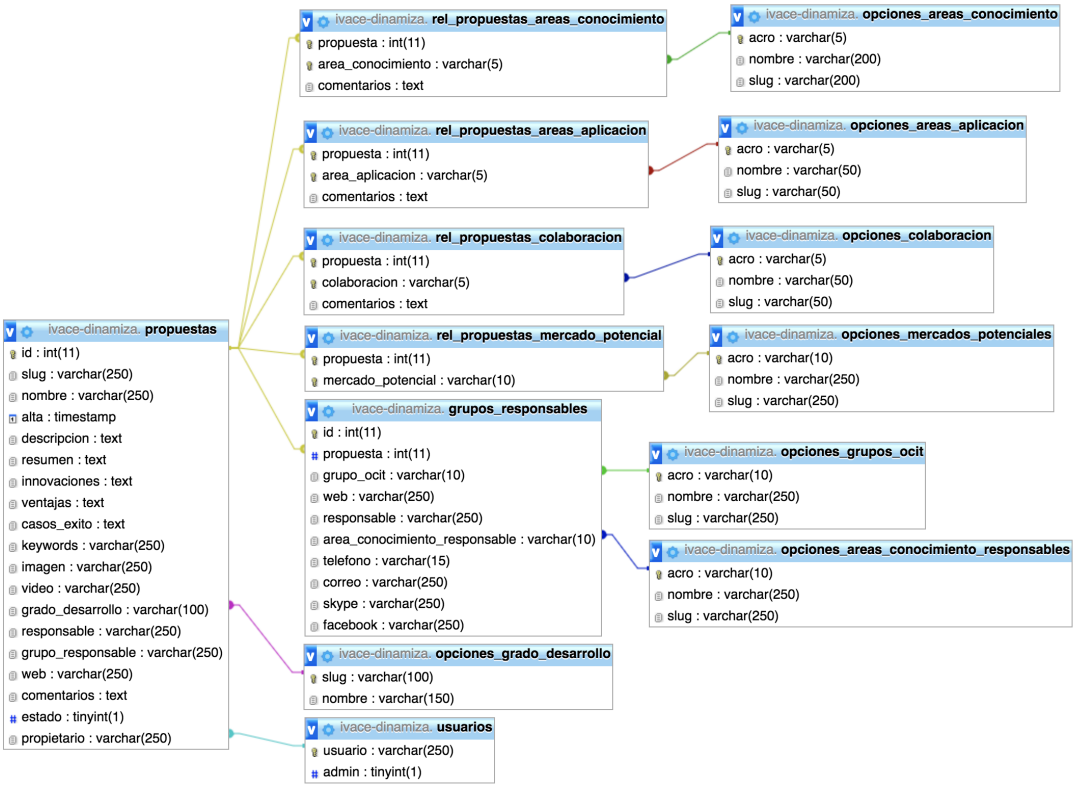

Figura 2.2: Diseño de la base de datos

La información se ha vinculado con una vista (Figura 2.3) que nos muestra una ficha explicativa de la solución con toda la información referente.

Figura 2.3: Información sobre una solución mostrada en la vista de ficha.

Además tiene un acceso directo (Figura 2.4) a contacto para más información sobre la solución y un enlace de descarga como PDF.

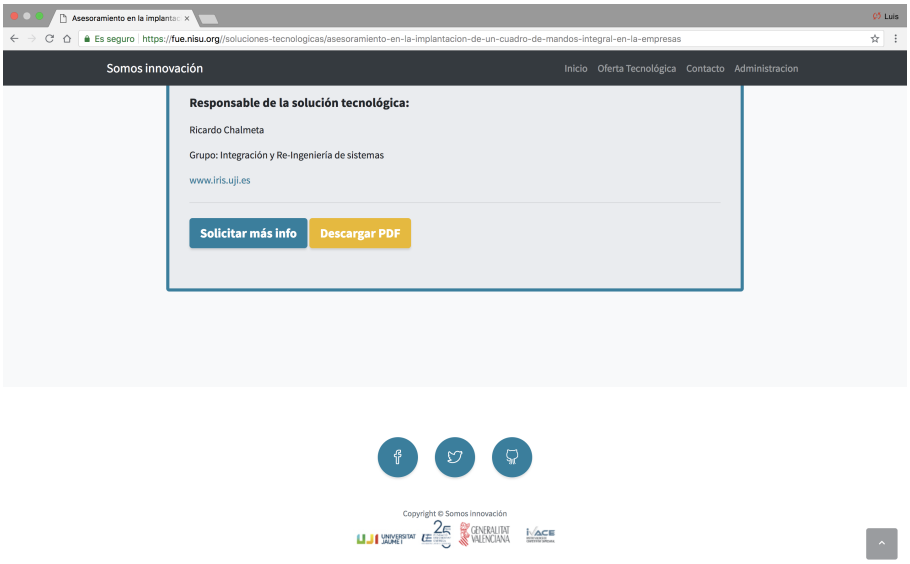

Figura 2.4: Acceso directo a acciones de contacto y descarga como PDF de la solución.

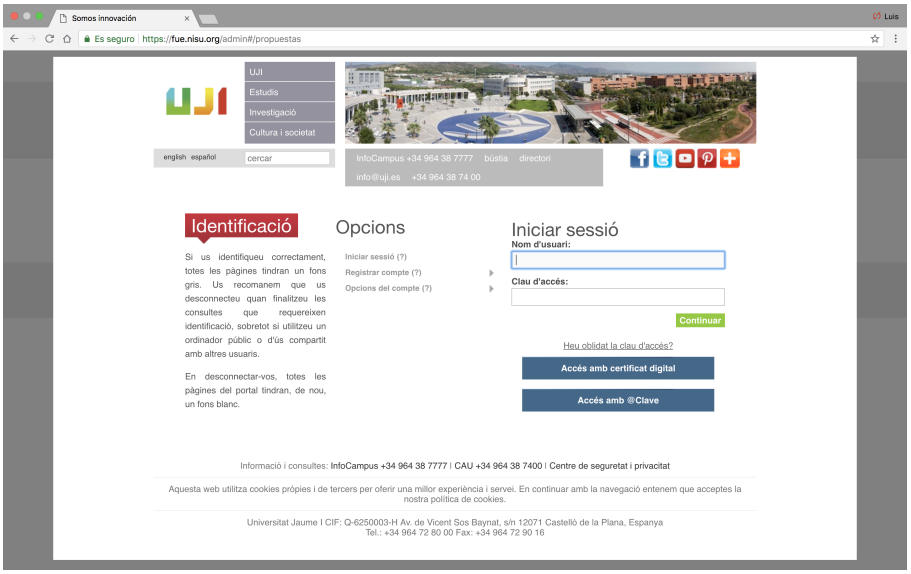

Figura 2.5: Inicio de sesión mediante el usuario de UJI.

Para el acceso al área de administración se ha vinculado el sistema de login con el sistema de usuarios existente en la UJI (Figura 2.5) y nos permite identificar a los profesores e investigadores sin necesidad de tener que volver a crear una cuenta en el nuevo sistema.

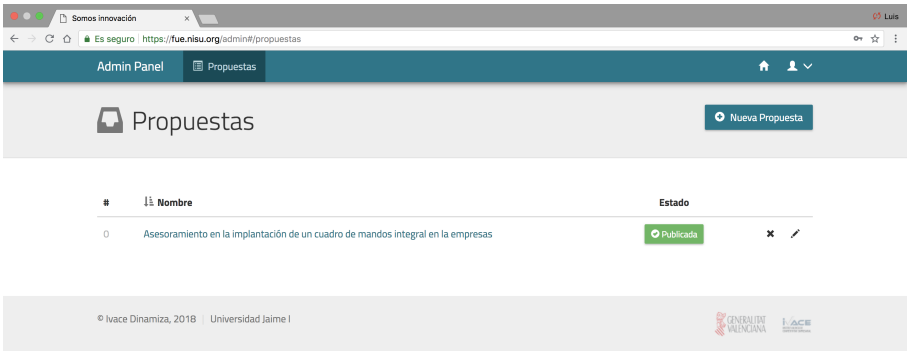

Figura 2.6: Panel principal de administración del portal.

El panel principal de administración (Figura 2.6) nos permite crear nuevas propuestas (soluciones), modificar o eliminar las ya creadas y consultar el estado de las soluciones que hayamos creado.

Cada solución pasará por tres estados diferente:

- **borrador**: cuando la propuesta todavía no se ha terminado de redactar (introducir toda la información).
- **pendiente de revisión**: cuando se ha enviado a los administradores del sistema para que se revise y se apruebe su publicación.
- **publicada**: cuando la propuesta se ha aceptado y se encuentra visible en el portal.

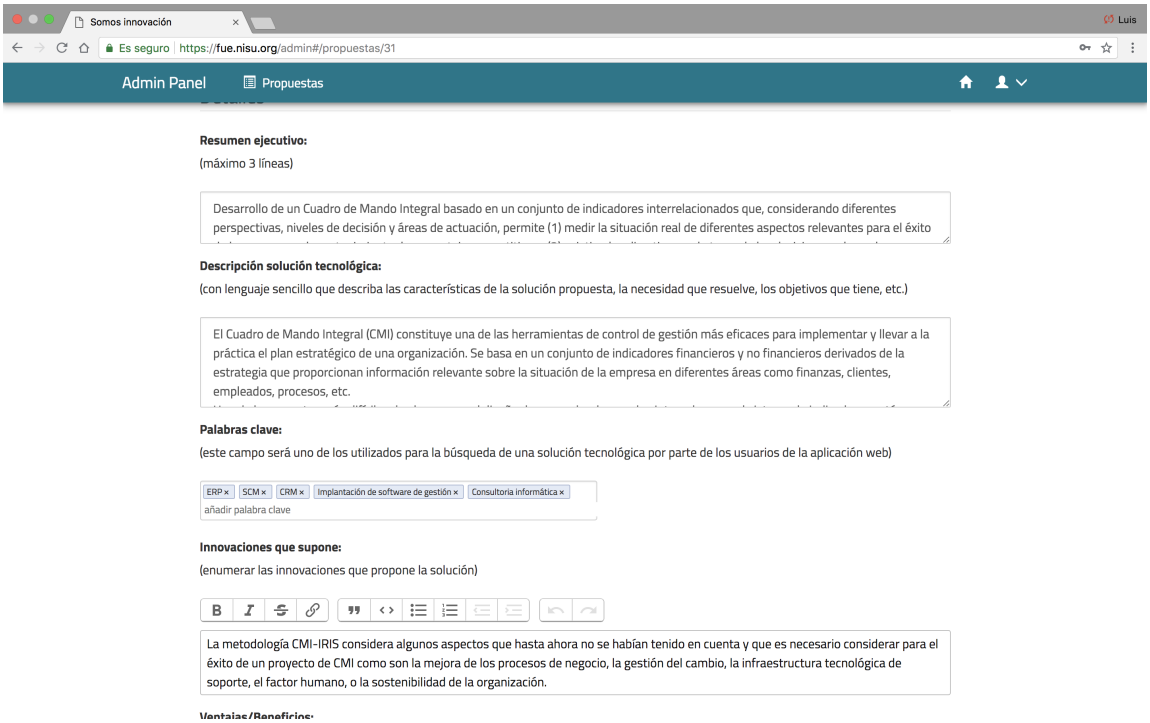

Figura 2.7: Ventana para la introducción de los datos sobre las propuestas (ofertas).

Finalmente, desde la ventana de introducción de los datos (Figura 2.7), los investigadores pueden rellenar la información sobre cada solución y elegir las categorías y taxonomías que clasificarán la solución para el buscador.

Paralelamente se ha desarrollado un bot utilizando phantomjs que es capaz de rastrear redes sociales que no permiten acceder a los datos de las publicaciones directamente desde su API. Phantomjs se trata de una libreria JavaScript que simula la interacción de un usuario real con un navegador.

```
//Step 2 - Rellenamos formulario de login
function () {
    console.log('[RUNTIME] Rellenando datos de login');
    page.evaluate(function () {
        document.getElementById("email").value = usuario;
        document.getElementById("pass").value = password;
        document.getElementById("loginbutton").click();
    });
    page.render('secuencia/1.png');
},
```

Figura 2.8: Login en Facebook utilizando bot phantomjs.

Como se puede apreciar (Figura 2.8), una vez ha navegado a la página de login, el bot es capaz de rellenar los campos de email y password hacer click en el botón de login como si de un usuario real se tratara.

```
//Step 4 - Visitamos Pagina de amigos y Scroll hasta el final
function () {
    console.log("[RUNTIME] Cargando amigos");
    page.open("https://www.facebook.com/profile.php?id=100014177133020&sk=friends", function (status) {
        console.log("[RUNTIME] Scroll hasta final de la pagina de amigos");
        var i = 1;
        loadInProgress = true;
        window.setInterval(function(){
            // page.render("secuencia/scroll" + i + ".png");
            var e = page.evaluate(function() { return document.getElementsByClassName("uiHeaderTitle")[3] });
            if(e == null){
                console.log("[RUNTIME] Scrolling -- ", i);
                page.evaluate(function(){
                    window.document.body.scrollTop = document.body.scrollHeight;
                });
                i += 1;
            } else {
                loadInProgress = false;
                return;
            }
        }, 2000);
    });
},
```

Figura 2.9: Simulado de scroll en muro de Facebook.

Finalmente, el bot es capaz de realizar scroll (Figura 2.9) en las páginas de muros de Facebook. Con las últimas actualizaciones de las redes sociales de tanto Twitter como Facebook, la

carga de nuevas publicaciones se realiza a medida que el usuario va haciendo scroll (va descendiendo en la página). Por tanto, el bot realiza dicho scroll simulando la acción de un usuario real y va capturando las nuevas publicaciones que la red social va cargando.

2.3.5. Conclusiones

La aplicación web desarrollada permite que lo usuarios (clientes) puedan realizar búsquedas por distintas vías de las soluciones que ofrecen los equipos de investigación de la universidad.

Además, permite que los investigadores puedan introducir de una forma rápida y sencilla nuevas ofertas de soluciones. Éstas ofertas, a continuación, pasan por un proceso de revisión y validación antes de ser publicadas en la web y puedan ser buscadas por los clientes.

Paralelamente, se ha desarrollado un bot rastreador capaz de simular el uso de una red social por parte de un usuario real para obtener información sobre las publicaciones.

El proyecto ha llegado a su fase final. Se están realizando pruebas con diferentes modelos capaces de clasificar los tuits en aquellos que pueden tratarse de usuarios que buscan alguna solución (clientes potenciales) y los que no. Posteriormente, se buscará el modelo óptimo y se automatizará el sistema de rastreo y análisis de la información.

Capítulo 3

Memoria TFG

3.1. Desarrollo del TFG

En esta sección vamos a desarrollar el fundamento teórico con el que hemos trabajado en este proyecto. En los siguientes apartados hablaremos sobre algoritmos de minería de datos que se pueden utilizar para clasificar y predecir el grado de satisfacción de los usuarios de una red social respecto a una persona o evento. Utilizaremos el software para computación estadística R [4] por tratarse de una herramienta multiplataforma que nos permitirá ejecutar el código en la mayoría de sistemas operativos.

La minería de datos se basa en la extracción de patrones útiles a partir de fuentes de datos sin conocimiento de su posible existencia. Es decir, los algoritmos de minería de datos nos permiten describir información por sí mismos, sin necesidad de ser dirigidos por las personas, guiados por los datos. [3] En el desarrollo del proyecto minaremos (analizaremos) diferentes tipos de información y buscaremos asociaciones entre los datos y patrones típicos de comportamiento.

Generalmente, se pueden minar dos tipos de patrones:

- Los patrones **descriptivos** que son aquellos que nos permiten describir las propiedades generales de un conjunto de datos, como los algoritmos de Descripción de clases y Conceptos, Análisis de Asociaciones y Clustering o Agrupación.

- Los patrones **predictivos** que nos permiten realizar inferencia a partir de un conjunto de datos para realizar predicciones basadas en datos históricos, como los algoritmos de Clasificación y Predicción o de Análisis de la evolución y desviación.

La minería de datos se basa en el aprendizaje computacional inductivo. Aprendizaje (inteligencia) porque mejora el comportamiento (rendimiento) identificando patrones en regularidades con los que es capaz de predecir futuras observaciones. Además intenta comprender la información y reducir la redundancia (*overfitting*) del que hablaremos más adelante.

Cualquier problema de aprendizaje inductivo se puede presentar de alguna de estas formas:

- Predictivos:

  Interpolación: generar a partir de un conjunto de $N+1$ puntos (datos) un polinomio P que represente el comportamiento de dichos puntos de forma que se pueda utilizar para predecir valores. Siendo X los valores de entrada, Y los valores de salida esperada. Generamos un polinomio de la siguiente forma:

  $$X(x_0, x_1, ..., x_N), Y(y_o, y_1, ..., y_N) \rightarrow$$

  $$\rightarrow P(X_x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + ... + b_n(x - x_0)...(x - x_{n-1})$$

  Tal que $P(X_x) = Y_x$, $\forall k = 0, ..., N$.

  Predicción Secuencial: Se trata de encontrar patrones estadísticamente relevantes en colecciones representadas de forma secuencial. Es decir, a partir de unas observaciones ordenadas secuencialmente se predice el siguiente valor de la secuencia. Un caso particular de *Interpolación* con 2 dimensiones, una discreta y otra continua.

  Aprendizaje Supervisado: tipo de aprendizaje computacional en el que se proporciona una serie de observaciones junto con la clase a la que pertenecen. El sistema aprende un clasificador que es capaz de predecir la clase de una nueva observación desconocida. Se trata de un caso particular de interpolación en el que la salida de la función (imagen) es discreta.

- Descriptivos:

  Aprendizaje No Supervisado: tipo de aprendizaje computacional en el que se proporcionan observaciones sin clases asociadas y se intenta detectar regularidades de cualquier tipo, como agrupaciones, contornos, valores anómalos o asociaciones.

Como vemos, con todas estas herramientas estadísticas y de minado de datos podemos extraer información muy valiosa de nuestros datos.

Hay que diferenciar claramente los siguientes conceptos: la Estadística tiene un enfoque mucho más teórico y realiza un énfasis en la validación de las hipótesis, sin embargo, el Aprendizaje automático tiene un enfoque más heurístico y realiza énfasis en la mejora del rendimiento del aprendizaje. La Minería de datos combina ambos con un enfoque que integra teoría y heurística

y que realiza un énfasis en el proceso completo de descubrimiento del conocimiento, desde la limpieza y el aprendizaje a la visualización final de la información minada.

En los siguientes apartados profundizaremos en algunas de las técnicas de minería de datos que nos servirán para analizar los tuits de los usuarios.

Comenzaremos buscando asociaciones en los datos que nos servirán para entrenar un modelo de inteligencia artificial capaz de clasificar los tuits en positivos y negativos.

Además obtendremos listas de palabras, con aquellas palabras positivas y negativas que mas aparecen, y calcularemos el porcentaje de tuits que han sido positivos y negativos respecto a cierto conjunto de tuits (de un *hashtag* o usuario). Aplicaremos también algoritmos de *clustering* (agrupamiento) para detectar los grupos (temas) de los que hablan los tuits y poder así analizar las temáticas más comentadas.

3.2. Análisis de Asociaciones

El Análisis de Asociaciones se trata de un algoritmo de aprendizaje no supervisado que [3] nos permite descubrir relaciones entre atributos (características de los valores de entrada) y valores (salidas) que suelen ocurrir frecuentemente juntos, por ejemplo, predecir con cierta probabilidad que los usuarios de género femenino y con edades comprendidas entre 20 a 30 años (atributos) publicarán un tuit negativo (valor) respecto a cierto tema.

Los **patrones frecuentes**, también llamados itemsets frecuentes, son patrones (conjuntos de items, valores y atributos) que aparecen normalmente juntos en un conjunto de transacciones. Se trata de encontrar items o atributos que tienden a ocurrir de forma conjunta dentro del conjunto de transacciones (consideramos cada transacción un tuit). A cada uno de los elementos que contiene una transacción lo llamaremos item y al conjunto de ellos itemset. Cada transacción puede estar formada por uno o diversos items, que en el caso de ser varios, cada subconjunto de ellos será un itemset distinto. Por ejemplo, la transacción $T = \{A, B, C\}$ está formada por 3 items (A, B, C) y sus posibles itemsets son: $\{A\}, \{B\}, \{C\}, \{A, B\}, \{B, C\}, \{A, C\}$ y $\{A, B, C\}$.

Por ejemplo, como veremos más adelante en la aproximación práctica, muchos de los tuits que analizamos contienen la palabra *dinero* o *viaje* asociada a *exconcejal*, *fundación* y *urbanismo*. Los itemsets, en este caso, serían las frases que contienen estas palabras. Cada palabra la consideramos un item. Y dentro de un itemset podemos encontrar subitemsets formados por varias palabras (atributos). Por lo tanto, para este ejemplo, las palabras (atributos) *exconcejal*, *fundación* y *urbanismo* se asocian con *dinero* o *viaje* (atributos). Y los valores serían la positividad o negatividad de los atributos.

La búsqueda de patrones frecuentes se trata de una acción fundamental a la hora de minar (analizar) correlaciones y relaciones entre los datos. Ayuda en la clasificación de los datos, clustering y otras tareas de minería de datos.

Sea $I = \{I_1, I_2, I_3, ...\}$ un itemset. Sea D un conjunto de transacciones en el que cada transacción T es un itemset no vacío de tal forma que $T \subseteq I$. A cada una de las transacciones se le asocia un identificador llamado TID. Sea A un conjunto de items, una transacción T se dice que contiene al conjunto A si $A \subseteq T$.

Definición 1 *Una **regla de asociación** es una implicación de la forma $A \to B$ donde $A \subset I$, $B \subset I$, $I \subset D$, $A \neq \emptyset$, $B \neq \emptyset$ y $A \cap B = \emptyset$. Dicha regla se cumple en un conjunto de transacciones D con un **support** s y una **confianza** c.*

Definición 2 *El **support** de un itemset formado por los subitemset $A \cup B$ es el porcentaje de transacciones en D tales que contienen $A \cup B \simeq P(A \cup B)$. Durante esta sección la notación $P()$ será el porcentaje de elementos en D que contengan $A \cup B$.*

$$sup(A \cup B) = \frac{\#(A \cup B)}{\#(D)} \simeq \frac{P(A \cup B)}{P(D)}$$

Siendo #(...) la cantidad de elementos de dicho itemset.

El **support** de una regla de asociación $A \to B$ se define como:

$$sup(A \to B) = sup(A \cup B) = \frac{\#(A \cup B)}{\#(D)}$$

Definición 3 *La **confianza** de una regla de asociación es el porcentaje de transacciones en D que contienen A y además contienen $B \simeq P(B|A)$.*

$$conf(A \to B) = \frac{sup(A \cup B)}{sup(A)} = \frac{\#(A \cup B)}{\#(A)} \simeq \frac{P(A \cup B)}{P(A)}$$

La **frecuencia de ocurrencia** de un itemset es la cantidad de transacciones que contienen a dicho itemset. El *support* de un itemset también se llama ***support* relativo**, mientras que la frecuencia de ocurrencia de un itemset también se llama ***support* absoluto**.

Veamos ahora un sencillo ejemplo para entender mejor el concepto de ***support*** y ***confianza***. Tenemos una muestra de 5 listas de la compra de personas diferentes en un supermercado:

P1 $\rightarrow$ { **pan**, **agua** }

P2 $\rightarrow$ { **agua**, huevos }

P3 $\rightarrow$ { leche }

P4 $\rightarrow$ { **pan**, **agua**, huevos }

P5 $\rightarrow$ { **agua** }

Vamos a intentar extraer alguna regla de asociación de ella.

El conjunto (itemset) total de items que tenemos es $I =$ { pan, agua, huevos, leche }.

El ***soporte*** de un subitemset formado por items de I es la proporción de personas (listas de la compra) que han comprado un conjunto de items en relación a la cantidad total de personas. Por ejemplo, el itemset $A =$ { pan, agua } tiene un soporte

$$sup(A) = \frac{2}{5} = 0{,}4$$

Entonces, el soporte es del 40 %, lo que quiere decir que 2 de cada 5 personas compran ambas cosas.

Ahora vamos a ver cuantas personas han comprado { huevos } además del itemset A.

Lo que estamos buscando es la **confianza** de la regla de asociación:

$A :$ { pan, agua } $\rightarrow B :$ { huevos }

$$conf(A \rightarrow B) = \frac{sup(A \cup B)}{sup(A)} = \frac{0{,}2}{0{,}4} = 0{,}5$$

Por lo tanto, la confianza de la regla de asociación es del 50 %, lo que quiere decir, que la regla será cierta en el 50 % de los casos.

Por esta razón, también se puede ver, como una probabilidad condicionada, $P(B|A)$, la probabilidad de encontrar una parte de la regla condicionada a encontrar la otra.

Definición 4 *Si el support relativo de un itemset I cumple con el support mínimo (es mayor) entonces diremos que I es un* ***itemset frecuente****.*

Definición 5 *Las reglas de asociación que satisfacen un support mínimo y una confianza mínima se llaman* ***asociaciones fuertes****.*

Por lo tanto, el minado (análisis) de reglas de asociación se puede reducir al minado de itemsets frecuentes que se realiza mediante el siguiente proceso:

1. **Búsqueda de itemsets frecuentes**: se trata de buscar todos aquellos itemsets que aparecen con una frecuencia mayor a la predeterminada por el *support* mínimo que hayamos fijado.
2. **Generación de reglas de asociación fuertes a partir de los itemsets frecuentes**: se trata de buscar aquellas reglas de asociación que satisfacen el *support* y confianza mínimos.

Este sencillo algoritmo, sin embargo, es computacionalmente bastante complejo en el primero de los pasos, la búsqueda de itemsets frecuentes, ya que conlleva la generación de muchas combinaciones de elementos. Para reducir el coste utilizaremos el siguiente teorema.

Teorema 1 *Si un itemset es frecuente $\rightarrow$ todos sus subsets también serán frecuentes.*

DEMOSTRACIÓN La demostración se sostiene a partir de la siguiente propiedad del *support*:

Sean I_1, I_2 dos itemsets.

$$\forall I_1, I_2 : (I_1 \subseteq I_2) \rightarrow sup(I_1) \geq sup(I_2)$$

El *support* de un itemset nunca excede el *support* de los subsets. Esta propiedad se conoce como anti-monotonía del *support*.

LEMA

Sea un k-itemset, un itemset de k elementos $\{a_1, a_2, ..., a_k\}$.

Este k-itemset contiene $\binom{k}{1} = k$ 1-itemsets $\{a_1\}, \{a_2\}, ..., \{a_k\}$.

$\binom{k}{2}$ 2-itemsets $\{a_1, a_2\}, ..., \{a_{k-1}, a_k\}$

...

La cantidad total de itemsets que contiene es $\binom{k}{1} + \binom{k}{2} + ... + \binom{k}{k} = 2^k - 1$.

La cantidad de itemsets (subsets) que contiene un k-itemset puede llegar a ser muy grande y difícil de procesar o guardar y necesitaremos introducir algunas definiciones.

Definición 6 *Si $X \subset Y$, es decir, que cada elemento de X está contenido en Y pero existe al menos un elemento de Y que no está contenido en X, diremos que Y es un* ***super-itemset*** *de X.*

Definición 7 *Un itemset X es* ***cerrado*** *en un conjunto de datos D si no existe un super-itemset Y tal que Y tenga el mismo support que X en D. Indicaremos que X es cerrado con $\bar{X}$*

$$\bar{X} \leftrightarrow \nexists Y : sup(Y) = sup(X) \text{ con } X, Y \in D \text{ y } X \subset Y$$

Definición 8 *Un itemset X es un* ***itemset cerrado frecuente*** *en un conjunto D si X es cerrado y frecuente en D.*

Definición 9 *Un itemset X es un* ***itemset frecuente maximal*** *en un conjunto D si X es frecuente y no existe un super-itemset Y tal que $X \subset Y$ y Y es frecuente en D.*

Ahora, podemos definir C como el conjunto de itemsets cerrados frecuentes de un conjunto de datos D que satisfacen cierto *support* mínimo. También M como el conjunto de itemsets frecuentes maximales de D que satisfacen cierto *support* mínimo. Entonces, C puede utilizarse para obtener el conjunto completo de itemsets frecuentes. Sin embargo, M solo almacena el *support* de los itemsets maximales, no contiene toda la información de sus correspondientes itemsets frecuentes. Para mejorar la generación de los candidatos a itemsets frecuentes podemos utilizar el algoritmo Apriori.

3.2.1. Algoritmo Apriori para generar candidatos confinados a itemsets frecuentes.

El algoritmo Apriori [13] para encontrar los itemsets frecuentes utiliza el conocimiento previo de los itemsets frecuentes en una aproximación iterativa conocida como búsqueda por niveles.

- *Primero*: Escaneamos la base de datos realizando un recuento para conocer la frecuencia de aparición de los elementos (1-itemsets) y seleccionando aquellos que tengan un *support* mayor que el mínimo. Obtendremos el conjunto de elementos frecuentes que llamamos L_1.
- *A continuación:* Utilizamos L_1 para encontrar L_2 que es el conjunto de 2-itemsets frecuentes. Con L_2 generamos L_3 y así sucesivamente hasta que no podamos encontrar más k-itemsets.

Para encontrar cada L_k necesitamos un escaneo completo de la base de datos y esto puede llegar a ser muy costoso. Para poder mejorar la eficiencia de la búsqueda de itemsets frecuentes podemos utilizar la propiedad Apriori para reducir el espacio de búsqueda.

PROPIEDAD Apriori

"Todos los subconjuntos de un itemset frecuente también son frecuentes"

Si un itemset I no tiene un *support* superior al mínimo, I no será frecuente.

$$sup(I) < minSupport$$

Si un item A se añade al itemset I, el itemset resultante $I \cup A$ no puede aparecer con más frecuencia que I. Por lo tanto tampoco será frecuente.

$$sup(I \cup A) \leq sup(I) < minSupport$$

Esta propiedad es antimonotona porque:

Si un itemset no es frecuente $\rightarrow$ Sus super-itemsets tampoco serán frecuentes.

DEMOSTRACIÓN

Utilizaremos L_{k-1} para encontrar los L_k para $K \geq 2$ realizando uniones y poda (generalización, eliminando nodos que conlleven un aumento del rendimiento sobre el conjunto de datos).

1. **Generación de candidatos con uniones**: para generar L_k uniremos los elementos de L_{k-1} entre ellos generando un conjunto C_k de candidatos. Sean I_1, $I_2 \in L_{k-1}$ dos itemsets. Asumiendo que los items están ordenados, es decir, $I_i[1] < I_i[2]$. La unión $L_{k-1} \bowtie L_{k-1}$ se realiza cuando los primeros $(k-2)$ elementos son comunes $\simeq (I_1[1] = I_2[1]) \wedge (I_1[2] = I_2[2]) \wedge ... \wedge (I_1[k-2] = I_2[k-2]) \wedge (I_1[k-1] < I_2[k-1])$. Esta última comprobación la haremos para evitar duplicados. El itemset generado por la union de $I_1 \bowtie I_2$ es $\{I_1[1], I_1[2], ..., I_1[k-2], I_1[k-1], I_2[k-1]\}$.
2. **Obtención de los itemsets frecuentes a partir de los candidatos**: Sea C_k el superitemset de L_k con items que pueden ser o no frecuentes, pero con todos los k-items frecuentes en C_k. Ahora, con una lectura de la base de datos podemos calcular la frecuencia de cada elemento de C_k y obtener L_k. Sin embargo, necesitamos reducir el tamaño de C_k ya que puede ser muy grande y para ello utilizaremos la propiedad Apriori. Cualquier $(k-1)$-itemset que no sea frecuente no puede ser un sub-itemset de un k-itemset frecuente. Por lo tanto, si cualquier $(k-1)$-sub-itemset de un k-itemset candidato no se encuentra en L_{k-1} entonces el candidato tampoco será frecuente y podremos eliminarlo de C_k, el conjunto de candidatos.

3.2.2. Generación de reglas de asociación

Una vez que hemos encontrado los itemsets frecuentes podemos aplicar la ecuación de confianza para obtener las reglas de asociación entre los itemsets con una confianza mínima. El *suport* ya lo hemos exigido a la frecuencia de los itemsets en el proceso de "poda".

$$confianza(A \rightarrow B) = P(B|A) = \frac{sup(A \cup B)}{sup(A)}$$

3.3. Algoritmos de Clustering (Agrupamiento)

Los algoritmos de clustering (agrupamiento) permiten juntar objetos similares en grupos. Un cluster es un conjunto de objetos similares entre ellos y desiguales a los objetos de otros grupos.

3.3.1. Algoritmos de Particionado

Los algoritmos de particionado permiten partir los conjuntos de datos en k grupos. Los principales se llaman *k-means* (3.3.1) y *k-medoids* ([5]).

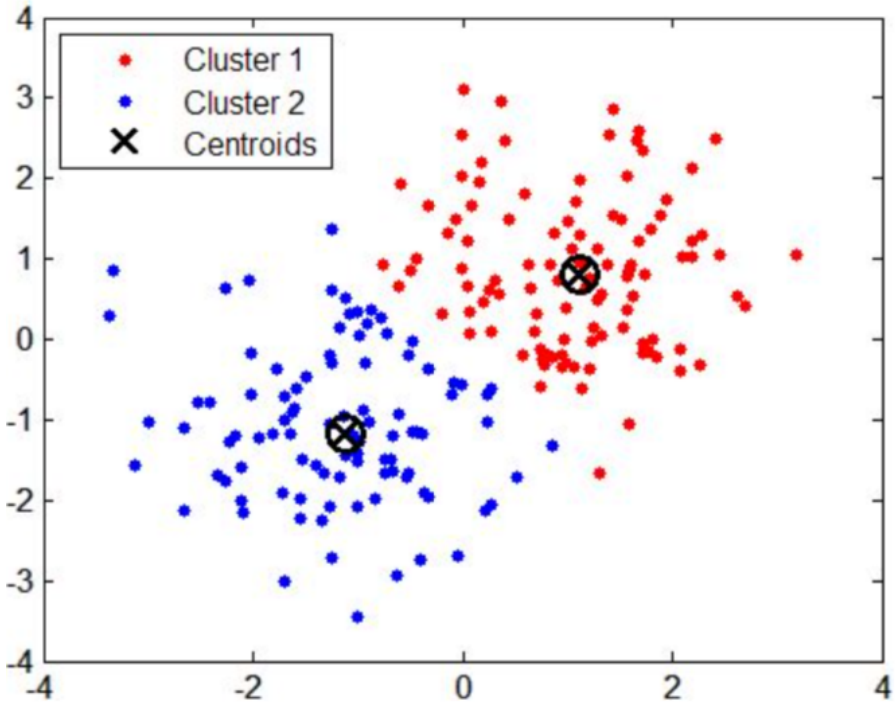

Figura 3.1: Ejemplo de algoritmo de particionado.

La similitud de los objetos de un grupo se mide con la distancia al centro de gravedad (centroide) del grupo al que pertence. Con variables categóriccas podemos utilizar tablas de similitudes o mapear las categorías a un intervalo. Con variables reales podemos utilizar una distancia métrica como la euclidea $d_E(P,Q) = \sqrt{(P_1-Q_1)^2+(P_2-Q_2)^2+\cdots+(P_n-Q_n)^2} = \sqrt{\sum_{i=1}^{n}(P_i-Q_i)^2}$ o la de Manhattan $d_1(P,Q) = ||P-Q||_1 = \sum_{i=1}^{n}|P_i-Q_i|$. Siendo P y Q dos vectores $P = (P_1, P_2, ..., P_n)$ y $Q = (Q_1, Q_2, ..., Q_n)$ de un espacio vectorial de dimensión n.

Algoritmo de particionado básico de *k-means*

1. Escogemos una cantidad k de centros para clústers aleatoriamente.

2. Asignamos cada objeto al clúster cuyo centro sea el más cercano usando una distancia métrica.

3. Recalculamos el centro (centroide) de cada clúster según los objetos asignados.

4. Repetimos los pasos 2 y 3 hasta converger, es decir, que los cambios en las asociaciones sean mínimas por debajo de un márgen.

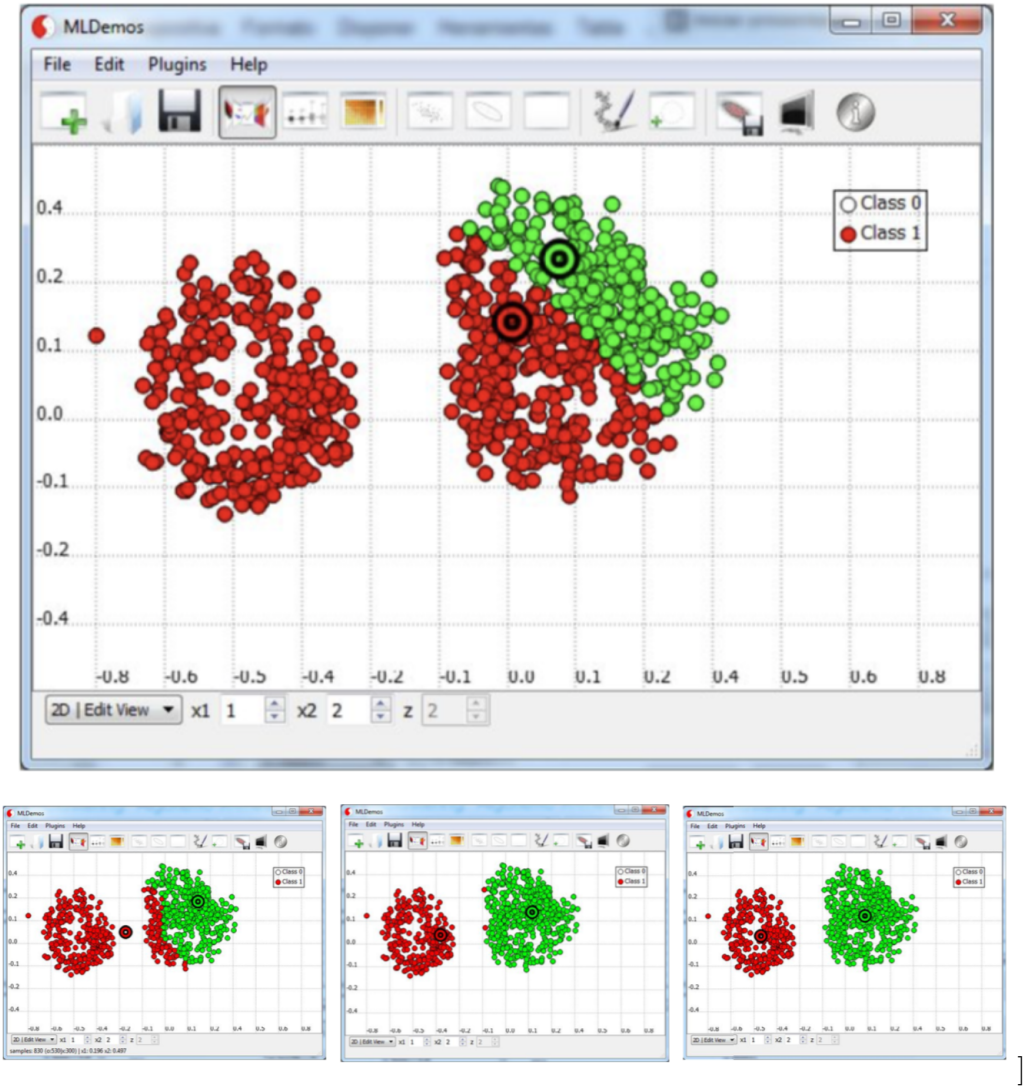

Figura 3.2: Proceso del algoritmo de particionado *k-means*.

Se trata de un algoritmo que funciona bien aunque es bastante sensible a los outliers (puntos que pertenecen a una clase pero están alejados del resto de puntos) y ruido y es poco escalable.

3.3.2. Algoritmos de Densidad

Estos algoritmos de particionado se basan en la densidad. Dos puntos pertenecen a un mismo clúster (grupo) si están conectados, es decir, si existe una alta densidad de otros puntos de la misma clase entre ellos. [1]

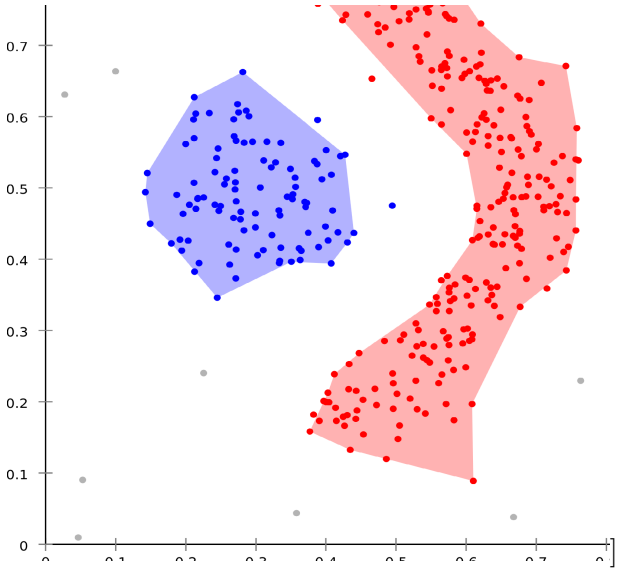

Figura 3.3: Ejemplo de algoritmo de densidad.

Algoritmo básico de densidad DBSCAN

1. Seleccionamos un punto p arbitrariamente.
2. Recuperamos todos los puntos conectados a p (alcanzables desde p).
3. Si p es un punto central (núcleo), se forma un clúster.
4. Si p es un punto frontera, DBSCAN visita el siguiente punto de la base de datos (conjunto de datos).
5. El proceso continua hasta que todos los puntos se hayan visitado.

Un punto p es un punto central o **núcleo** si al menos $minPuntos$ puntos se encuentran a una distancia ϵ de el punto p y, además, esos puntos son directamente alcanzables desde p.

Un punto q es alcanzable desde p si existe una secuencia de puntos $p_1, ..., p_n$ tal que $p_1 = p$ y $p_n = q$, donde cada punto p_{i+1} es directamente alcanzable desde p_i. Es decir, todos los puntos de la secuencia son núcleos (puntos centrales) con la posible excepción de q.

Si p es un núcleo (punto central), éste punto p forma un clúster junto a otros puntos (núcleo o no) que sean alcanzables desde él. Cada clúster, entonces, contiene al menos un punto núcleo (central).

El algoritmo visita varias veces los puntos como candidatos a varios clústers. Es de utilidad utilizar un índice para almacenar los vecinos de cada punto y calcular una matriz de distancias al principio para evitar el recálculo de las distancias.

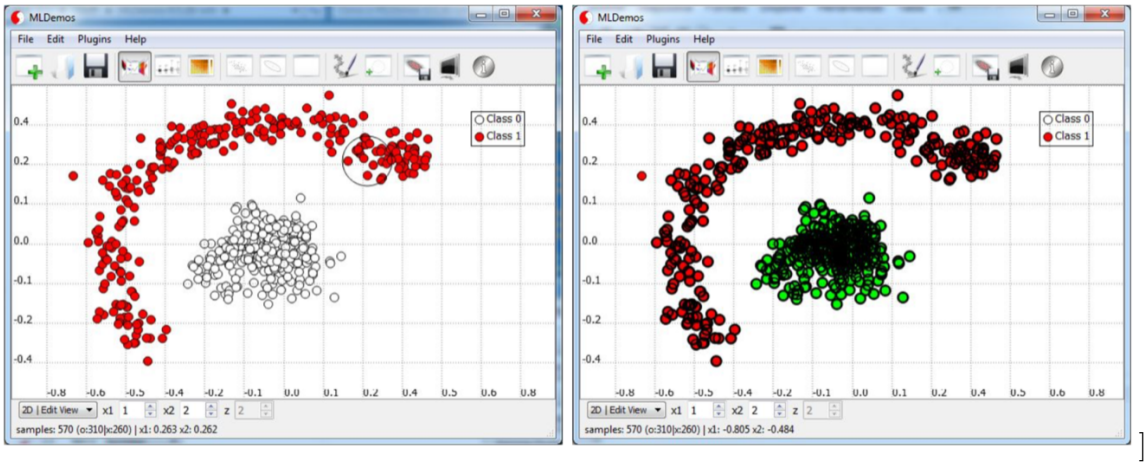

Figura 3.4: Ejemplo del proceso del algoritmo DBSCAN.

3.3.3. Algoritmos Jerárquicos

Los algoritmos de agrupación jerárquicos proporcionan dendogramas (diagramas en forma de árbol) que representan relaciones entre grupos anidados.

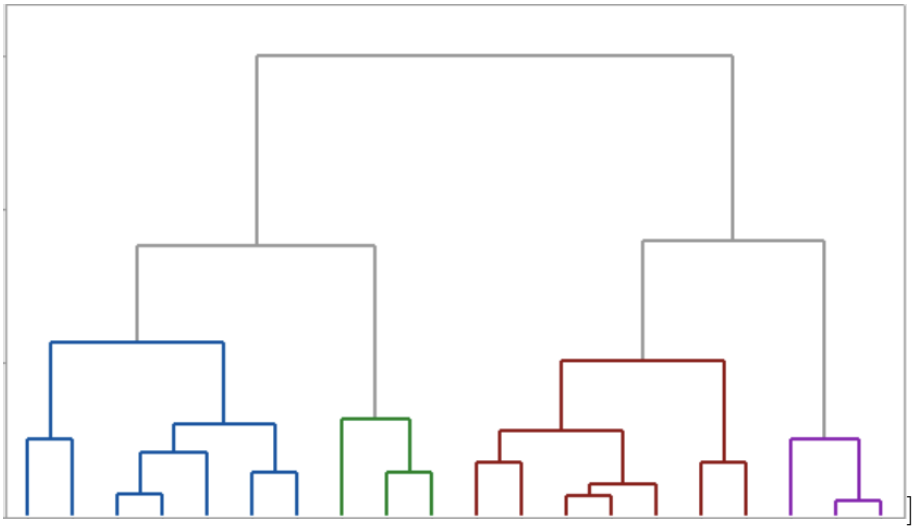

Figura 3.5: Ejemplo de dendograma generado después de aplicar un algoritmo jerárquico.

Se pueden realizar dos aproximaciones al algoritmo:

- **Aglomerativa**: empezando creando un clúster para cada objeto y agregando entre los clústers.
- **Divisivo**: empezando con un único clúster con todos los objetos y dividiendo recursivamente.

Algoritmo de agrupación jerárquico AGNES

1. Asignamos a cada objeto un clúster, de forma que para N objetos tendremos N clústers.
2. Encontramos el par de clústers más similar y los fusionamos. Utilizamos una métrica para medir la distancia entre clústers como medida de similitud. Además, podemos definir la distancia de distintas formas:

   (NEAREST NEIGHBOUR DISTANCE): La menor distancia entre los individuos del grupo y el individuo exterior.

(FURTHEST NEIGHBOUR DISTANCE): La mayor distancia entre los individuos del grupo y el individuo exterior.

(DISTANCIA ENTRE CENTROIDES): La distancia entre un centroide de un grupo y otro centroide de otro grupo.

3. Calculamos las distancias entre el nuevo clúster y cada uno de los otros.
4. Repetimos los pasos 2 y 3 hasta que todos los objetos se agrupen en un clúster dentro de cierto margen (utilizado para medir la similitud).

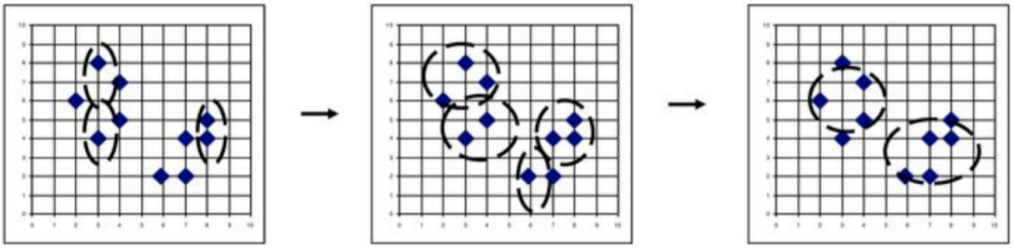

Figura 3.6: Ejemplo del proceso del algoritmo jerárquico AGNES.

Una vez procesadas todas las publicaciones podemos elegir el algoritmo más apropiado para agrupar (clustering) en categorías (temas) las publicaciones de los usuarios y poder detectar así las temáticas más comentadas por los usuarios.

3.4. Algoritmos de Clasificación y Predicción

Los algoritmos de Clasificación y Predicción [3] nos permiten generar modelos que son capaces de agrupar los datos en clases con el objetivo de predecir la clase a la que pertenece un objeto desconocido. Esto es útil para, por ejemplo, clasificar cada uno de los tuits en positivos, negativos o neutrales según el contenido de los mismos. Realizamos una pequeña introducción a los algoritmos básicos y profundizaremos en las estructuras en forma de árbol.

3.4.1. Naïve Bayes

Se trata de una implementación clásica relativamente sencilla que permite obtener buenos resultados. Uno de los métodos más utilizados en la minería de textos basado en la teoría estadística Bayesiana, el clasificador que se obtiene se conoce como Clasificador Bayesiano.

Asumiendo que todos los atributos son independientes, se basa en obtener la probabilidad de que cierto elemento E_n pertenezca a una determinada clase C_i. Siendo A_k cada uno de los k atributos de un elemento.

$$P(C_i|E_n) = P(C_i) \cdot \prod_k P(A_k|C_i)$$

Este algoritmo modela las relaciones probabilísticas entre el conjunto de atributos de una variable y la clase a la que pertenece. [2]

Definición 10 *La **probabilidad condicional** de que una variable aleatoria pueda tomar un valor particular dado el valor de otra variable aleatoria $P(Y = y|X = x)$ hace referencia a la probabilidad de que una variable Y pueda tomar el valor y dado que la otra variable X tome el valor x.*

Las probabilidades condicionales de X e Y están relacionadas por:

$$P(X, Y) = P(Y|X)P(X) = P(X|Y)P(Y)$$

Que se relaciona con el Teorema de Bayes

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

En esta notación consideramos X al conjunto de atributos A_k e Y a la clase C_i a la que pertenece. Volviendo a la notación anterior y en un supuesto 'Naïve' en que todos los atributos son independientes, entonces:

$$P(A_1, A_2, ..., A_k|C_i) = P(A_1|C_i) \cdot ... \cdot P(A_k|C_i) = \prod_k P(A_k|C_i)$$

- *Si el atributo es categórico $P(A_k|C_i)$ se calcula como la frecuencia relativa de elementos que contienen el valor A_k como atributo de clase C_i:*

$$P(A_k|C_i) = \frac{\#\ elementos\ de\ clase\ C_i\ con\ el\ atributo\ = A_k}{\#\ elementos\ de\ clase\ C_i}$$

- *Si el atributo es continuo $P(A_k|C_i)$ se estima utilizando la función de densidad de Gauss. Se calcula la media $\mu_{i,k}$ y la desviación estándar $\sigma_{i,k}$ para cada atributo A_k utilizando*

datos de la clase C_i y se obtiene:

$$P(A_k|C_i) = \frac{1}{\sqrt{2\pi}\sigma_{i,k}} e^{-\frac{(A_k-\mu_{i,k})^2}{2\sigma_{i,k}^2}}$$

3.4.2. SVC y SVM - *Support Vector Classifier* y *Support Vector Machine*

Éste método, también conocido como Maquina de Soporte Vectorial [10], se trata también de uno de los métodos más utilizados para clasificación y análisis de sentimientos.

Realizaremos una aproximación a los *Support Vector Classifier* que nos ayudará a entender como funciona un *Support Vector Machine*.

Considerando cada uno de los valores de entrada con sus atributos como un vector $(A_1, A_2, ..., A_k)$. Podemos representar cada uno de estos valores (vectores) en un espacio k-dimensional. Veamos un simple ejemplo utilizando solo 1 atributo, en un espacio de dimensión 1.

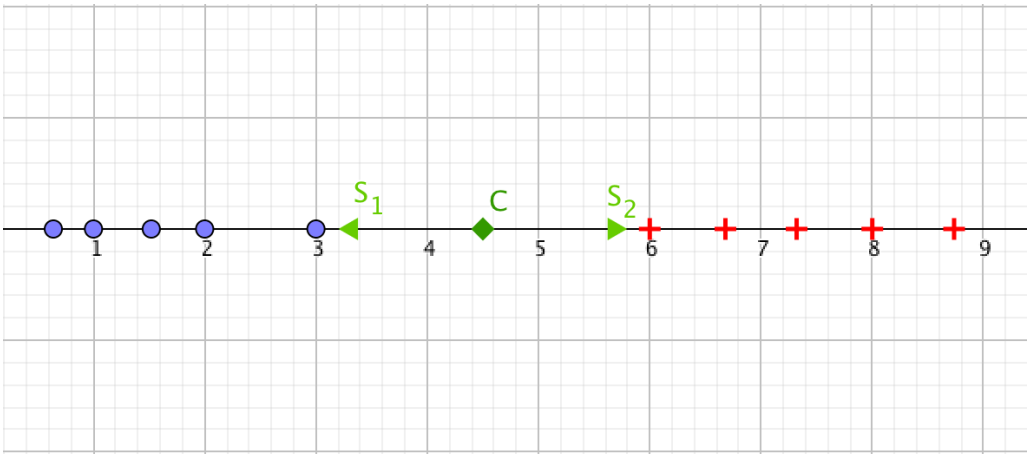

Figura 3.7: Ejemplo de SVM para espacio 1-dimensional

En este ejemplo, la recta representa todos los posibles valores de A_1, la primera de las dimensiones del vector de valores de entrada, ya que estamos trabajando con 1 sola dimensión.

Estamos introduciendo el funcionamiento básico del *support vector classifier*. Sin embargo, el funcionamiento del *support vector machine* va mucho más allá; representa cada uno de los valores de entrenamiento como puntos en el espacio y genera un modelo capaz de clasificar cada

uno de los puntos utilizando clasificadores. Además, son capaces de realizar clasificaciones no lineales utilizando kernels, que también introducimos, más adelante en este apartado.

El funcionamiento básico de un clasificador se basa en encontrar un hiperplano C que separe los vectores de atributos que representan cada una de las publicaciones en dos grupos, intentando que esta separación sea la máxima posible. Para ello, utiliza dos vectores llamados vectores de soporte S_1, S_2 que se encargan de definir los márgenes de dicha separación.

Si ahora consideramos 2 atributos, en un espacio 2-dimensional conseguimos una representación similar a la siguiente. El hiperplano que separa los grupos se trata de una línea por tratarse de un espacio de 2 dimensiones y los vectores de soporte son líneas paralelas al hiperplano separador.

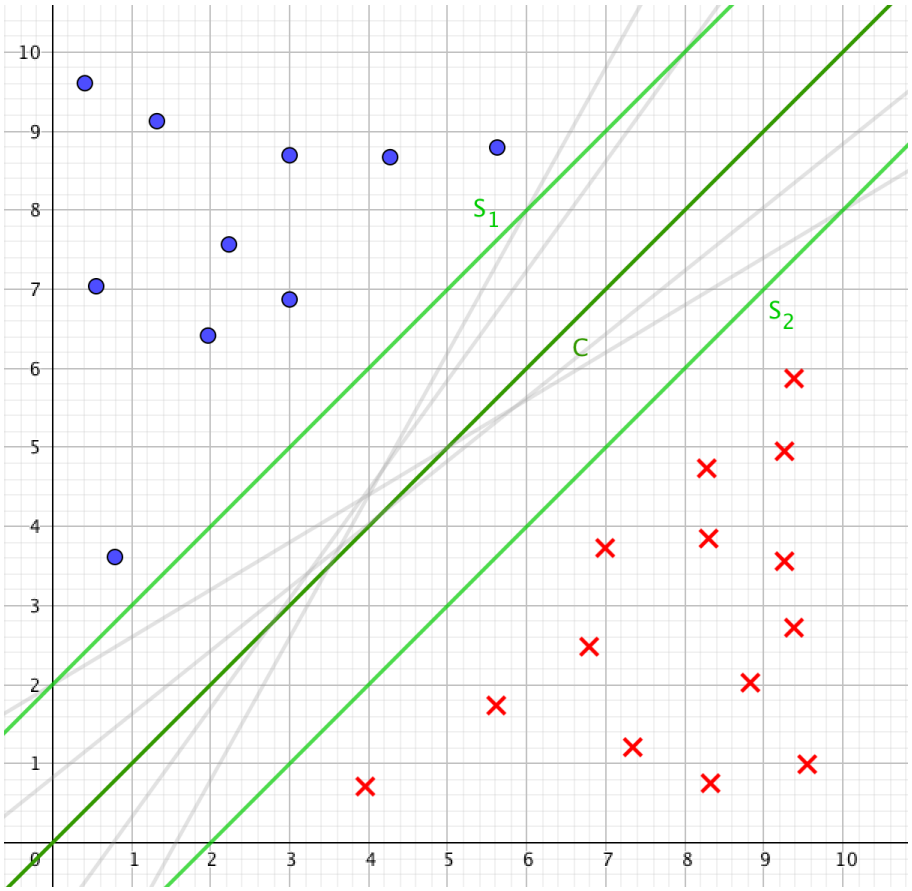

Figura 3.8: Ejemplo de SVM para espacio 2-dimensional

Como se puede apreciar en color gris, existen infinitos hiperplanos que nos permiten separar

en dos clases los valores. Sin embargo, se trata de encontrar aquel que permita un margen máximo entre los elementos de las dos categorias; con la distancia entre los vectores de soporte y el hiperplano separador (margen) la máxima posible.

Sea $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ un conjunto de valores de entrada con su clasificación [10] , donde $x_i \in R^k$ e $y_i \in \{-1, +1\}$ podemos definir un hiperplano separador como la siguiente función lineal:

$$C(x) = (w_1 x_1 + ... + w_k x_k) + b =< w, x > +b$$

Siendo w y b coeficientes reales que se pueden calcular a partir de la siguiente restricción:

$$y_i(< w, x_i > +b) \geq 0 \text{ con } i = 1, ..., k$$

Sin embargo, a partir de la ecuación anterior se pueden obtener infinitos planos. Se trata de encontrar el que maximice el margen entre el hiperplano separador y los vectores de support. Llamamos a esta distancia (márgen) τ y se puede calcular a partir de la ecuación de distancia entre un hiperplano $D(x')$ y un elemento x': $\frac{|D(x')|}{||w||}$. Por lo tanto, todos los elementos cumplirán además:

$$\frac{y_i D(x_i)}{||w||} \geq \tau \text{ con } i = 1, ..., n$$

Se trata de encontrar el valor de w que maximice el margen. Para ello, la escala del producto de τ y la norma de w se fija, de forma arbitraria, a la unidad $T||w|| = 1$. Llegando a la conclusión final de que aumentar el márgen es equivalente a disminuir la norma de w. Expresando la expresión anterior de esta forma $T = \frac{1}{||w||}$ se observa que encontrar un hiperplano separador óptimo se trata de enccontrar el que posee un máximo margen, es decir, un valor mínimo de $||w||$.

Kernel

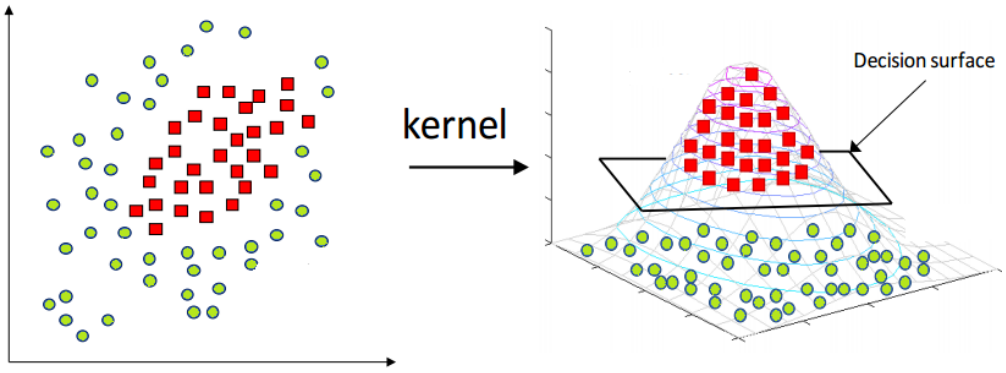

Figura 3.9: Ejemplo de kernel

No en todos los casos es posible utilizar lineas o planos separadores y se necesita utilizar una región no-lineal para separar las clases. Mediante el uso de un ***kernel*** se mapea los datos a un espacio vectorial distinto en el que un hiperplano lineal se puede utilizar para separar las clases. La función kernel transforma los datos a una dimensión superior en el que una separación lineal es posible.

3.4.3. Estructuras en forma de árbol

Los árboles de decisión son estructuras en forma de árbol que dadas una serie de variables como entrada son capaces de predecir la salida o clasificar los datos. Extraen información característica de los valores de entrada y realizan una partición del espacio de valores de entrada de forma que pueden luego encajar las nuevas observaciones desconocidas prediciendo la clase a la que pertenecen.

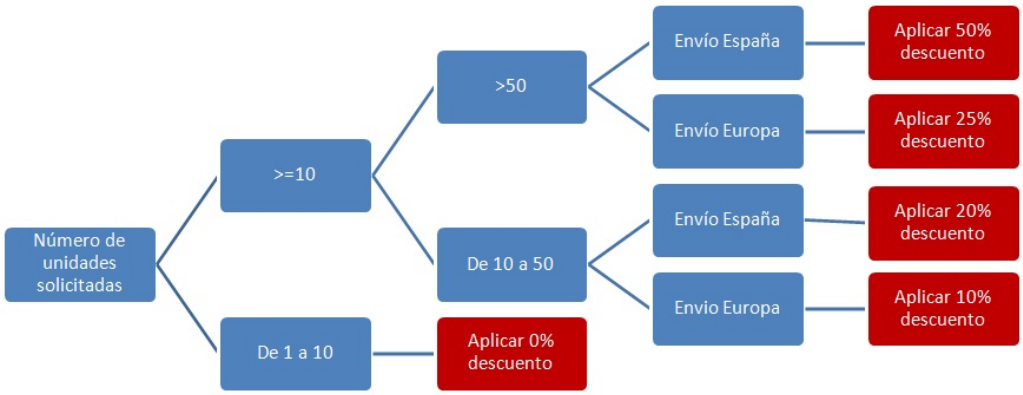

Figura 3.10: Ejemplo de árbol de decisión [8]

Los arboles de decisión son estructuras en forma de árbol con un solo nodo raíz en los que cada nodo representa un test sobre un atributo, cada rama representa un resultado del test y cada hoja una clase.

Para clasificar cada dato se debe recorrer el árbol de decisión desde la raíz hasta las hojas. Con este proceso conseguimos, además, la información del proceso de razonamiento (nodos) seguidos, es decir, la justificación de las respuestas.

Sea x_1, x_2 dos valores de entrada. Sea Y la salida o respuesta. [11] En cada partición podemos modelar Y con un valor distinto. Podemos escoger el punto de partición para conseguir que nuestro modelo se aproxime el máximo a la salida deseada. Cada partición, a su vez, se puede dividir en más subespacios hasta que cierta condición de parada se aplique. Al terminar la partición obtenemos:

- M regiones, siendo M la cantidad de regiones en que hemos dividido el espacio de valores de entrada.
- $\{R_1, R_2, ..., R_M\}$ las regiones en las que lo hemos dividido el espacio de valores de entrada.
- C_m la constante o valor de salida para cada una de las regiones R_m.

Con todo esto, siendo $I(x_1, x_2, ..., x_m)$ la región R_m a la que pertenece un valor de entrada,

podemos predecir la salida Y dados C_m en una R_m de la siguiente forma:

$$f(X) = \sum_{m=1}^{M} C_m I\{(x_1, x_2, ...x_m) \in R_m\}$$

El proceso de generación o entrenamiento de un árbol de decisión se basa en la elección de los atributos que más información nos proporcionen y sus valores para realizar las divisiones del espacio de valores de entrada y clasificarlos para obtener una salida.

Definición 11 *La **cantidad de información** que contiene un elemento (item, palabra o itemset, conjunto de palabras) se relaciona con la probabilidad de que aparezca ese elemento. Si el elemento aparece con mayor frecuencia tendrá menor cantidad de información para nuestro árbol. Sea X un elemento de los valores de entrada.*

$$\#informacion(X = x) = -log(P(X = x))$$

Definición 12 *La **entropía** es la media de cantidad de información que podemos obtener para cierta pregunta. También se puede ver como la cantidad de incertidumbre (único desde un punto de vista de clasificación) que tiene un elemento de dicha región.*

$$H(X = x) = \sum P(X = x)(-log(P(X = x))$$

Definición 13 *La **ganancia de información** es la diferencia de entropía entre antes y después de realizar una pregunta (antes de realizar una partición). Sea q la pregunta o criterio seleccionado para realizar la partición.*

$$Ganancia = H(x) - H(x|q)$$

Se divide en subespacios basándose en algún atributo que nos de como resultado los subespacios más puros o diferenciados. Para decidir los puntos que marcarán las divisiones, las "preguntas", queremos minimizar la suma de los cuadrados $\sum(y_i - f(x_i))^2$, siendo $f(x_i)$ el valor predicho para el valor de entrada x_i y siendo y_i el valor real de salida (correcto) para dicha entrada. Utilizaremos el valor de salida $c_m = promedio(y_i|x_i \in R_m)$ igual al promedio de las salidas de los elementos de dicha región.

Computacionalmente no podemos considerar todas las posibles particiones. Realizamos una aproximación a corto plazo de arriba a abajo de forma recursiva, llamada particionado binario.

- *de arriba a abajo*: comenzamos en la parte superior del árbol, donde todas las observaciones pertenecen a una misma región y, a continuación, sucesivamente vamos particionando el espacio de predicciones. Cada partición inicia dos ramas hacia abajo en el árbol.
- *a corto plazo*: en cada paso se elige la mejor partición sin continuar y comprobar cual sería mejor en el siguiente paso.

Sea j la variable para realizar el particionado. Sea s el punto de particionado. Podemos definir un par de semi-planos $R_1(j,s) = \{x|x_j \leq s\}$ y $R_2(j,s) = \{x|x_j > s\}$. Buscaremos los valores de j y s que minimicen

$$min_{j,s}[min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2]$$

con $c_1, c_2 = y_{\hat{R}_1}, y_{\hat{R}_2}$, la media de valores de dicha región.

Para cada variable candidata al particionado, el punto o valor donde realizar la partición se puede encontrar leyendo todas las entradas. Por lo tanto, es factible encontrar el valor de (j,s). Sin embargo, realizar este proceso puede ser computacionalmente costoso.

Una vez hemos encontrado el punto de particionado, realizamos la partición en dos nuevas regiones, y repetiremos el proceso de particionado con las dos nuevas regiones que hemos generado y el resto de regiones hasta encontrar una condición de final de carrera. Es importante tener en cuenta que un árbol demasiado grande podría ajustarse demasiado a los datos de entrada y dar malos resultados para datos diferentes, y un árbol demasiado pequeño podría no detectar las estructuras internas de los datos.

Existen varios algoritmos de particionado para árboles. Primero, introduciremos el algoritmo básico y en el siguiente apartado hablaremos de otros algoritmos con fines más precisos.

ALGORITMO de particionado básico de un Árbol de decisión

1. Empezamos con un único nodo con todos los datos.
2. Si todos los datos son de la misma clase se trata de un nodo hoja.
3. Si todos los datos NO son de la misma clase, seleccionamos el atributo que mejor separe los datos en clases distintas:

   Creamos una rama para cada posible valor del atributo seleccionado.

   Ejecutamos recursivamente con cada una de las ramas que hemos creado.

Algoritmos de particionado para Árboles

Existen distintos algoritmos de particionado. A la hora de dividir las regiones de los valores de entrada, el atributo y valor que se eligen, dependen de la métrica que mide si los subespacios resultantes serán puros o homogéneos; depende de la medida que se utilice para comparar los valores o regiones obtenidos y saber si estarán más aislados o menos. Podemos utilizar algunos algoritmos con fines más precisos:

- **ID3 - C4.5**: Estos algoritmos buscan conseguir la mayor cantidad de información en cada etapa del proceso. Comparan la cantidad de información que consiguen con cada uno de los atributos y eligen el que la maximiza. También lo podemos ver como el que más reduce la entropía. Sea p la cantidad de elementos clasificados correctamente. Sea n la cantidad de elementos incorrectos. Sea d la cantidad total de elementos. Se utiliza la siguiente medida:

$$-(\frac{p}{d})log_2(\frac{p}{d}) - (\frac{n}{d})log_2(\frac{n}{d})$$

  C4.5 es una mejora de ID3 que funciona con variables continuas y categóricas. Transforma los elementos continuos en regiones discretas estableciendo un umbral para cada región.

- **Cart**: Se puede utilizar para arboles de regresión y clasificación. Tiene el objetivo de conseguir la menor cantidad de Gini [1] en cada etapa del proceso, es decir, minimizar la probabilidad de falsos positivos 'podando' una rama del árbol en un atributo sin permitir que continue.

- **CHAID**: (Detector Automatico de Interacciones Chi-Cuadrado). Este algoritmo mira las variables de entrada e identifica las posibles interacciones entre las variables. Utiliza chi-cuadrado x^2 para medir la correlación entre las variables independientes y las clases; para decidir si la diferencia entre una variable predictiva y la clase es suficientemente significativa. Si lo es, se divide el nodo en base a dicha variable. Mide la discrepancia entre una distribución observada y otra teórica, también llamada bondad de ajuste. Sea x_i una observación. Sea y_i una salida esperada (clase asociada). La prueba x^2 de Pearson se mide con:

$$x^2 = \sum_i \frac{(x_1 - y_i)^2}{y_i}$$

Una vez hemos encontrado el atributo de las variables de entrada que nos interesa utilizar después de aplicar un algoritmo (criterio) de particionado, crearemos una rama del árbol para cada uno de los valores que pueda tomar dicho atributo según la clase a que pertenece. Sea x_1 una de las nuevas ramas (nodo o región) tal que $x_1 \subseteq x$ el nodo raíz. Sea A_1 el atributo elegido

[1] Se explica en el siguiente apartado *Medidas del error para árboles de clasificación.*

para la partición. Sea v_1 uno de los valores que puede tomar dicho atributo. Para dicho valor del atributo obtenemos la región $x_1 = \{x \in X : I(A_1) = v_1\}$.

El principal problema es el ***overfitting***. Éste fenómeno se produce cuando el modelo se ajusta demasiado a los datos utilizados para el entrenamiento, luego no funciona correctamente con otros datos. La solución es realizar un 'podado' al árbol reemplazando algunos nodos por hojas. Encontrar un límite entre lo que es útil o no, entre un modelo demasiado lineal o uno que sea demasiado ajustado (un polinomio de un grado muy alto).

Sea $T \subset T_0$ un subárbol que se puede obtener podando T_0. Sea el nodo m que define una región R_m. Sea $|T|$ el tamaño del subárbol, la cantidad de nodos hojas o regiones que tiene. Sea N_m la cantidad de elementos $\{x_i \in R_m\}$ que pertenecen a una región R_m. Sea C_m el valor constante asignado a la salida de una región $\hat{C_m} = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$. Y procedemos de la misma forma con el resto de valores y atributos, dividiendo el espacio de valores de entrada.

Definición 14 *Podemos definir un* ***criterio de complejidad de costos****, que es el criterio que queremos minimizar a la hora de generar un subárbol realizando una poda. Utilizamos un parámetro $\alpha \geq 0$ para ajustar la proporción entre el tamaño del árbol y su precisión para ajustarse a los datos.*

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{C_m})^2$$

$$C_\alpha = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$$

Con $\alpha = 0$ obtenemos T_0 el árbol completo, y cuando aumentamos el valor las ramas del árbol se van 'podando'. Para cada α existe un subárbol T_α que minimiza $C_\alpha(T)$.

Para encontrar los subárboles T_α utilizamos un método conocido como ***poda del enlace más débil*** en el que iterativamente vamos 'podando' los nodos internos que producen el menor aumento por nodo en $\sum_m N_m Q_m(T)$ hasta producir un único nodo padre.

Con esto obtenemos una secuencia finita de subárboles que deberá contener T_α. Para obtener el valor de α utilizaremos un método conocido como ***k-fold cross-validation***. Con éste método podemos encontrar el valor de $\hat{\alpha}$ que minimice la suma de cuadrados de los valores *cross-validated* hasta encontrar el subárbol final $T_{\hat{\alpha}}$. Hablaremos de éste método en los siguientes apartados, pero antes vamos a introducir algunas medidas del error.

3.4.4. Medidas del error para árboles de clasificación

Cuando trabajamos con árboles de clasificación no podemos utilizar la suma de los cuadrados como medida del error para hacer las particiones. Una alternativa es el **ratio de error**. Intentamos asignar a un valor de entrada en una región concreta un valor de salida igual al valor que aparece más veces en dicha región como salida.

Definición 15 *El **ratio de error de clasificación** es la fracción de elementos en una región que no pertenecen a dicha región.*

Sea un nodo m representando una región R_m con N_m observaciones.

La proporción de elementos de tipo k en el nodo m es:

$$\hat{p_{mk}} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

Para clasificar una observación en el nodo m asignamos la clase $K(m) = argmax_k(\hat{p}_{mk})$, la clase que aparece mayoritariamente en el nodo m.

Para los árboles de regresión podemos utilizar la suma de los cuadrados de los errores de la medida de impureza $Q_m(T)$ con los nodos pero para clasificación no podemos utilizar ésto. Vamos a introducir otras medidas de impureza $Q_m(T)$ para árboles de clasificación:

El ***misclassification error*** o error de clasificación no es lo suficiente sensitivo a los cambios en el tamaño del árbol. Sea y_i la clase a la que pertenece un elemento.

$$E = \frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq K(m)) = 1 - \hat{p}_{mk}(m)$$

Para ello utilizaremos otras medidas como el ***Gini*** y la ***Cross-entropy*** o desviación.

Definición 16 *El **Índice de Gini** mide la variación total en cada una de las clases k. Toma un valor pequeño si todos los $\hat{p}_{mk}$ son cercanos a 0 o 1. Mide la pureza, si un nodo contiene mayoritariamente elementos de una sola clase.*

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = \sum_{k \neq k'} \hat{p}_{mk}\hat{p}_{mk'}$$

Definición 17 *La **cross-entropy** o desviación toma valores cercanos a 0 si $\hat{p}_{mk}$ es cercano a 0 o 1. Toma valores pequeños si el nodo es puro.*

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk})$$

Como $0 \leq \hat{p}_{mk} \leq 1$, entonces $0 \leq -\hat{p}_{mk} log(\hat{p}_{mk})$.

Para evaluar la calidad de una partición concreta será útil el ***Gini*** o ***cross-entropy***, pero si queremos medir la calidad de las predicciones en general del árbol después del proceso de 'poda', es mejor utilizar el ***misclassification error*** o ratio de error de clasificación.

3.4.5. Cross-validation y otras mejoras

Cross-validation es una técnica para seleccionar el mejor modelo. Un modelo que funcione bien con los datos que ha visto y los que no ha visto. Un modelo que no se ajuste demasiado (***overfitting***) adquiriendo algunos patrones de los datos que no son realmente importantes y que producen una clasificación errónea en otros datos de entrada.

La clave está en utilizar diferentes *data sets* para entrenar el árbol y comprobar si funciona bien con todos los conjuntos de datos. Separaremos algunos datos para realizar pruebas en el modelo y crearemos múltiples conjuntos de datos para entrenamiento que sean hijos del conjunto de entrenamiento original. ***Cross-validation*** busca el mejor rendimiento de media con todos los conjuntos de datos de entrenamiento. El mejor modelo será aquel que tenga un mejor rendimiento de media con todos los conjuntos de datos.

K-FOLD CROSS VALIDATION

1. Dividimos el conjunto de datos de entrenamiento en k subsets iguales D_k.

2. Utilizamos cada uno de los subconjuntos D_k para tests y para entrenamiento hasta que cada uno de los subconjuntos se haya utilizado al menos 1 vez para tests.

Normalmente, un división $k = 10$ es lo más utilizado.

LEAVE 1 OUT

1. Elegimos aleatoriamente un subconjunto para tests y utilizamos el resto para entrenamiento.
2. Realizamos esto con todos los datos, utilizando cada elemento para pruebas y el resto para entrenamiento.

LEAVE P OUT

1. Elegimos aleatoriamente p elementos que utilizamos para tests y el resto para entrenamiento.
2. Realizamos esto con todos los datos, hasta que todas las posibles combinaciones de p elementos se han utilizado para tests. Pueden haberse utilizado los elementos varias veces para tests en distintos conjuntos de datos de prueba.

MONTECARLO

Con este método se divide aleatoriamente los subconjuntos de entrenamiento y tests y no se garantiza que cada uno de los elementos se haya utilizado al menos una vez.

Los algoritmos de ***cross-validation*** nos permiten elegir entre varios algoritmos, ajustar parámetros o identificar los elementos que son relevantes. Nos ayudan a evitar el *overfitting* pero no completamente, necesitaríamos conocer el modelo y esto es imposible.

3.4.6. Ensemble Learning - Bosques de decisión

Los modelos tienden a padecer cierto *overfitting* pero podemos combatirlo realizando una aproximación diferente al problema. Podemos entrenar diferentes modelos de forma que el *overfitting* de cada uno de ellos se puede compensar con el de los otros modelos.

La salida del modelo resultante, realizado a partir de la combinación de diversos modelos que han podido o no ser entrenados con los mismos datos o parámetros, se puede decidir:

- Eligiendo la salida mayoritaria de todos los modelos.
- Calculando la media de la salida de todos los modelos.
- Otorgando pesos (dando mas importancia) a las salidas de los modelos.

Podemos utilizar diferentes técnicas para generar los conjuntos de datos de entrenamiento para los *ensemble* (conjuntos de árboles):

- ***BAGGING (Bootstrap Aggregating)***: cada modelo del *ensemble* se entrena con un conjunto de entrenamiento distinto que se genera a partir de un conjunto de entrenamiento inicial. Se asigna a cada modelo un mismo peso y se elige como salida el voto de la mayoría. Los conjuntos de entrenamiento se generan utilizando una técnica conocida como *Bootstrap Sampling* en la que todos los elementos tienen la misma probabilidad y se produce reemplazamiento.
- ***BOOSTING***: se añade modelos iterativamente al *ensemble* y en cada iteración se genera un conjunto de datos de entrenamiento en el que los elementos que no se habían clasificado correctamente tienen más peso. De este modo, los modelos que funcionaban mal por sí solos se pueden combinar con otros modelos creando un modelo mucho más inteligente, que cubra los defectos que tenían por separado.

  ADABOOST: es una mejora de *Boosting* que utiliza pesos en las salidas de los modelos e iterativamente va añadiendo y definiendo los pesos óptimos para cada etapa.

Definición 18 *El proceso de* ***Stacking*** *consiste en entrenar un modelo para aprender a combinar los resultados de los modelos simples y decidir los pesos para cada salida. También se conoce como AKA Blending o AKA Stacked Generalization.*

Capítulo 4

Aplicación Práctica con R y Resultados

4.1. Configuración y Preprocesado de los tuits

Vamos a extraer los datos de Twitter a través de su API con R Studio y necesitaremos algunos paquetes instalados:

- `Devtools`: nos permite conectarnos a GitHub para cargar el código de paquetes almacenados en dicho sistema de control de versiones.
- `rjson`: nos permite traducir los datos en formato JSON procedentes de la API de Twitter a un formato manejable por R.
- `bit64`: nos permite almacenar enteros por encima de 2^{63} para poder trabajar con un gran volumen de información sin perder precisión y con un mejor rendimiento.
- `httr`: necesario para poder realizar peticiones HTTP a la API de Twitter y que funcione correctamente el paquete twitteR.
- `twitteR`: nos permite importar la información deseada de Twitter a nuestra sesión de R.
- `udpipe`: este paquete de NLP (programación neurolingüística) será el encargado de tokenizar (extraer los lexemas y clases gramaticales) y anotar el texto extraído de los tuits [9].
- `plyr`: nos permite separar las palabras de los tuits para poder realizar el análisis de sentimientos.

- `syuzhet`: paquete que nos permite realizar un análisis de sentimientos de un texto, a partir de diccionarios preentrenados por el Nebraska Literary Lab en varios idiomas [6].
- `ggplot2`: nos permite generar distintas visualizaciones gráficas de los resultados categorizados.
- `RColorBrewer`: nos permite personalizar los colores de las visualizaciones gráficas.
- `httpuv`: utilizado para poder realizar el `handshake` con la fuente de datos web.
- `RCurl`: necesario para poder realizar peticiones Curl al servidor de datos.

El creador de GitHub y del paquete `twitteR`, Jeff Gentry, ha preparado el paquete para realizar las peticiones a Twitter para que utilice el sistema `OAuth handshake` en las peticiones que realiza. Para poder utilizar este sistema de autentificación, primero deberemos crear una `app` en Twitter desde `https://apps.twitter.com` y obtener así las claves de la API y los tokens de seguridad.

Primero, cargamos lo paquetes y librerías auxiliares necesarias y a continuación, configuramos las credenciales para acceder a la API de Twitter:

```
1 api_key <- "API_KEY"
2 api_secret <- "API_SECRET"
3 ac_token <- "ACCESS_TOKEN"
4 ac_token_secret <- "ACCESS_TOKEN_SECRET"
5 setup_twitter_oauth(api_key, api_secret, ac_token, ac_token_secret)
```

Ahora ya tenemos acceso a la API y podemos comenzar a extraer los datos.

Para esta aplicación, a modo de ejemplo, realizaremos una búsqueda utilizando el hashtag #primariasPP recogiendo 8180 tuits publicados entre el 28-06-2018 a las 23:27:33 y el 08-07-2018 a las 22:31:11.

Una vez realizada una búsqueda es muy apropiado realizar un preprocesado de los datos para eliminar elementos (palabras) que no nos aportan información como los hashtag, menciones o urls. Para realizar dichas tareas utilizaremos algunas expresiones regulares:

```
1 # Eliminamos URLs
2 tuits.df_limpio <- gsub("http[[:alnum:]]*", "", tuits.df)
3 # Eliminamos RT
4 tuits.df_limpio <- gsub("RT_@[a-z,A-Z]*:_","", tuits.df_limpio)
5 # Eliminamos hashtags #
6 tuits.df_limpio <- gsub("#[a-z,A-Z]*","", tuits.df_limpio)
```

```
7 # Eliminamos menciones - @usuario
8 tuits.df_limpio <- gsub("@[a-z,A-Z]*","", tuits.df_limpio)
```

Nos interesará tener los tuits, en algunos casos, con el formato original y, en otros casos, limpios de forma que no contengan palabras que puedan alterar los resultados finales.

4.2. Análisis de Asociaciones y Cálculo de tablas de frecuencias

Las herramientas de programación neuronlingüística analizan las palabras y de alguna forma 'entienden' el texto de entrada por el contexto. Clasifican cada uno de los términos en sustantivos, adjetivos, verbos, etc. según el resto de palabras que rodean y contextualizan la información. En este caso utilizaremos los tuits sin limpiar considerando los hashtags y menciones también información relevante, ya que también nos interesa conocer cuales son los hashtags que mas se han utilizado y cuales son los usuarios que mas se ha mencionado. Utilizaremos la libreria *udpipe* de análisis lingüístico ya que contiene varios modelos preentrenados para español y nos facilitará el proceso de análisis del texto.

El primer paso es realizar un análisis UPOS a todos los tuits y sus palabras. El análisis UPOS - Universal Parts of Speech (partes del habla universales) consiste en asignar a cada palabra una categoría que identifica la funcionalidad sintáctica (sustantivos, adjetivos, verbos, etc). Se extrae información sintáctica y se reemplazan las palabras por sus lemmas (forma conceptual abstracta de una palabra que representa un significado específico, pero no tiene ninguna conjugación específica).

```
tuits_analizados <- udpipe_annotate(udmodel, x = tuits$text,
                                    doc_id = tuits$id)
tuits_analizados <- as.data.frame(tuits_analizados)
```

Y obtenemos una tabla con todo el texto de los tuits etiquetado como el siguiente:

	token_id	token	lemma	upos	xpos	feats	head_token_id	d
Los políticos Pablo C...	6	Los	el	DET	NA	Definite=Def|Gender=Masc|Number=Plur|PronTyp...	7	c
Los políticos Pablo C...	7	políticos	político	PROPN	NA	Gender=Masc|Number=Plur	4	a
Los políticos Pablo C...	8	Pablo	pablo	PROPN	NA	Gender=Masc|Number=Sing	7	a
Los políticos Pablo C...	9	Casado	casado	PROPN	NA	NA	8	f
Los políticos Pablo C...	10	y	y	CCONJ	NA	NA	11	c
Los políticos Pablo C...	11	José	josé	PROPN	NA	NA	8	c
Los políticos Pablo C...	12	Ramón	ramón	PROPN	NA	NA	11	f
Los políticos Pablo C...	13	García	garcía	PROPN	NA	NA	11	f
Los políticos Pablo C...	14	Hernández	hernández	PROPN	NA	NA	11	f
Los políticos Pablo C...	15	son	ser	VERB	NA	Mood=Ind|Number=Plur|Person=3|Tense=Pres|Ve...	18	c
Los políticos Pablo C...	16	los	el	DET	NA	Definite=Def|Gender=Masc|Number=Plur|PronTyp...	18	c
Los políticos Pablo C...	17	mejores	mejor	ADJ	NA	Degree=Cmp|Number=Plur	18	a
Los políticos Pablo C...	18	posicionados	posicionado	NOUN	NA	Gender=Masc|Number=Plur|VerbForm=Part	1	p
Los políticos Pablo C...	19	en	en	ADP	NA	NA	21	c
Los políticos Pablo C...	20	la	el	DET	NA	Definite=Def|Gender=Fem|Number=Sing|PronType...	21	c
Los políticos Pablo C...	21	defensa	defensa	NOUN	NA	Number=Sing	18	n

Figura 4.1: Ejemplo de texto después de análisis UPOS

Una vez tenemos el texto etiquetado, es decir, tenemos los lemmas de cada una de las palabras y la clase sintáctica a la que pertenecen, podemos calcular las tablas de frecuencia para cada categoría sintáctica:

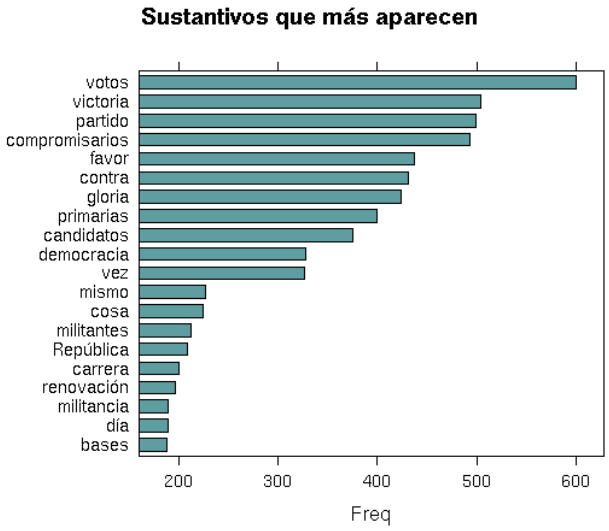

Figura 4.2: Análisis de frecuencias de aparición para sustantivos de *#primariasPP*.

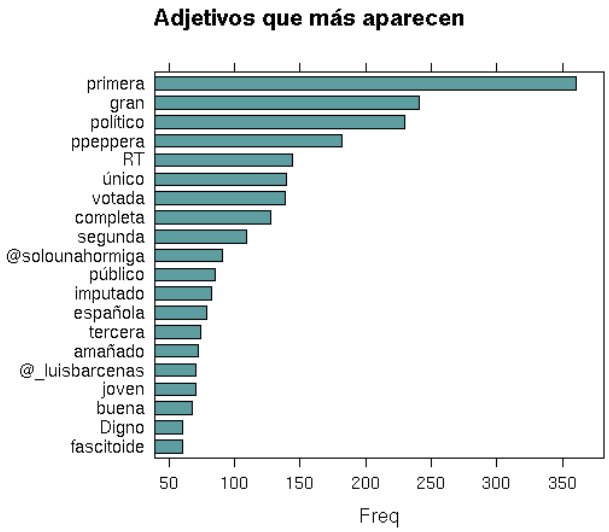

Figura 4.3: Análisis de frecuencias de aparición para adjetivos de *#primariasPP*.

La libreria udpipe utiliza ***MorphoDiTa***. Se trata de un diccionario morfológico y etiquetador capaz de analizar las frases y realizar una clasificación sintáctica de los términos. Utilizamos un modelo que ha sido prentrenado para el idioma español. En las gráficas algunos términos como RT (retuits, publicaciones republicadas por otros usuarios) y las menciones se han interpretado como adjetivos.

También utilizamos el algoritmo RAKE (Rapid Automatic Keyword Extractor). Un método no supervisado e independiente del idioma para extraer palabras clave (keywords) y analizar su frecuencia de aparición en el conjunto de tuits.

Las palabras clave que extaemos son los itemsets frecuentes. Parejas de palabras (items) que aparecen con mayor frecuencia. Para ello utilizamos algoritmos que aplican la teoría que introducimos en 3.2.

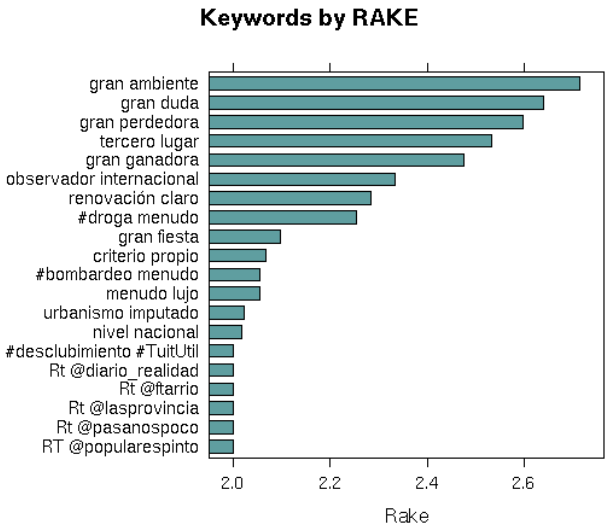

Figura 4.4: Análisis de frecuencias de aparición con algoritmo RAKE de *#primariasPP*.

Éste algoritmo utiliza una lista de stopwords (palabras como articulos o preposiciones) junto a los simbolos de puntuación para separar las frases en posibles palabras clave candidatas. A continuación utiliza tablas de ocurrencia de los términos y tablas de coocurrencia de pares de términos para calcular un ránking de las palabras mas mencionadas (extraer keywords).

En este caso, para nuestro propósito, el algoritmo RAKE intenta extraer las keywords; las palabras más importantes del conjunto y que son también las que mejor describen las diferentes temáticas de las que han hablado los usuarios.

	term1	term2	cooc
874602	mantra	ppeppero	0.9876727
80645	@jla	#MociónDeCensura	0.9825509
530428	favor	gloria	0.9818788
457227	dinero	exconcejal	0.9765149
517427	dinero	fundación	0.9765149
1103302	dinero	urbanismo	0.9765149
970854	anciano	residencia	0.9757877
873058	base	ppepperar	0.9756411
486176	contra	favor	0.9734313
886902	@lsfernandeza	presidentar	0.9717695
1129176	exconcejal	viaje	0.9708945
1129232	fundación	viaje	0.9708945
1129777	urbanismo	viaje	0.9708945
1114920	aspirante	vehículo	0.9700309
589452	dinero	imputado	0.9648356
277359	@compolitico	comunicativo	0.9635688
451509	@compolitico	estrategia	0.9635688
965088	ppepperar	renovación	0.9628341
966431	.@vox_	representante	0.9608398
1129299	imputado	viaje	0.9592796
945077	balcones.	Recordad	0.9573722
354756	.@vox_	derecha	0.9571641

Figura 4.5: Análisis de frecuencias de ocurrencia de términos emparejados de *#primariasPP*.

En esta última tabla podemos ver los términos que suelen aparecer emparejados y el porcentaje de veces que aparecen juntos cuando al menos uno de los dos aparece. Esta aplicación es un ejemplo de búsqueda de itemsets frecuentes de dimensión 2. Para ello se ha utilizado una implementación del algoritmo Apriori 3.2.1.

4.3. Análisis de sentimientos

En esta sección realizaremos una aproximación práctica al análisis de sentimientos. El objetivo principal del análisis de sentimientos será el de poder clasificar los tuits que analicemos en positivos y negativos.

Primero realizaremos una comparación entre los diferentes algoritmos de clasificación y predicción, que vimos teóricamente en la sección 3.4 sobre un modelo que entrenaremos. Trabajaremos con Naïve Bayes, SVM, Árboles de Decisión y Bosques de Decisión.

A continuación, utilizaremos un lexicon (modelo preentrenado) en español para analizar texto extraído de tuits publicados en Twitter con un cierto hashtag.

4.3.1. Comparativa de Algoritmos

Para realizar pruebas con los diferentes algoritmos de clasificación y predicción necesitamos entrenar un modelo de cada tipo. Para realizar el entrenamiento, utilizaremos un conjunto de frases que etiquetaremos positiva o negativamente. Además, este conjunto de frases nos servirá para realizar tests sobre el modelo que entrenemos.

En este caso utilizaremos frases en inglés por la mayor facilidad y versatilidad que nos ofrece este lenguaje; ya que la gramática es más sencilla y los verbos no tienen tantas conjugaciones será más fácil extraer información útil de los textos.

Utilizaremos en R las librerías de RTextTools y e1071 [7] que se utilizan para Clasificación de Texto Automática con Aprendizaje Supervisado y contienen todos los algoritmos que hemos mencionado durante el desarrollo de la parte teórica.

Conjunto de entrenamiento y tests

```
pos_tuits =  rbind(
        c('I love this house', 'positive'),
        c('This view is amazing', 'positive'),
        c('She is my best friend', 'positive'),
        c('I feel great this morning', 'positive'),
        c('I am so excited about the event', 'positive')
)
```

```
neg_tuits = rbind(
        c('I do not like this house', 'negative'),
        c('This view is horrible', 'negative'),
        c('She is my enemy', 'negative'),
        c('I feel tired this morning', 'negative'),
        c('I am not looking forward to the event', 'negative')
)

test_tuits = rbind(
        c('feel happy this morning', 'positive'),
        c('larry friend', 'positive'),
        c('not like that man', 'negative'),
        c('house not great', 'negative'),
        c('your song annoying', 'negative')
)
```

Para generar el conjunto de frases de entrenamiento y tests utilizamos frases con sentidos opuestos para los conjuntos de positivos y negativos, y para los elementos de test introducimos palabras relacionadas con las de los tests y otras que no hemos entrenado.

Esto nos permitirá medir el desempeño de los distintos algoritmos sobre un conjunto de datos similar al que nos encontraremos en Twitter.

Tenemos por tanto, una muestra de 5 tuits (*test tuits*) que queremos clasificar como positivos o negativos, a partir de una muestra de 10 tuits en la que hemos etiquetado 5 tuits como positivos y otros 5 como negativos.

Lo primero que necesitamos hacer ahora es generar una matriz con cada uno de los términos y marcar la aparición en los distintos tuits. Esta matriz se llama **matriz de términos - documentos** y se utiliza frecuentemente para tareas de indexación y búsqueda (Google). Necesitamos tener una representación explícita del documento; normalmente se representa de forma vectorial, donde cada componente (dimensión) del vector representa cada una de las palabras (términos) del léxico, y el valor asignado a cada componente corresponde a la frecuencia de aparición dentro de cada documento (tuit en este caso).

Por lo tanto, tendremos una matriz (4.6) en la que cada columna representará una palabra y cada fila representará uno de los tuits.

	about	amazing	annoying	best	enemy	event	excited	feel	forward
I love this house	0	0	0	0	0	0	0	0	0
This view is amazing	0	1	0	0	0	0	0	0	0
She is my best friend	0	0	0	1	0	0	0	0	0
I feel great this morning	0	0	0	0	0	0	0	1	0
I am so excited about the event	1	0	0	0	0	1	1	0	0
I do not like this house	0	0	0	0	0	0	0	0	0
This view is horrible	0	0	0	0	0	0	0	0	0
She is my enemy	0	0	0	0	1	0	0	0	0
I feel tired this morning	0	0	0	0	0	0	0	1	0
I am not looking forward to the event	0	0	0	0	0	1	0	0	1
feel happy this morning	0	0	0	0	0	0	0	1	0
larry friend	0	0	0	0	0	0	0	0	0
not like that man	0	0	0	0	0	0	0	0	0
house not great	0	0	0	0	0	0	0	0	0
your song annoying	0	0	1	0	0	0	0	0	0

Figura 4.6: Muestra de la Matriz de términos documentos. Total 15 filas y 29 columnas.

La matriz generada tiene 15 filas representando los tuits y 29 columnas representando cada una de las palabras. Como os podéis imaginar, si los tuits son más largos y aumentamos la cantidad de 15 tuits, el tamaño (columnas) de la matriz se disparará rápidamente como veremos más adelante.

Pruebas con Naïve Bayes

```
> table(tuits[11:15, 2], predicted)
          predicted
           negative positive
  negative        2        1
  positive        0        2
```

Figura 4.7: Resultados del test con el algoritmo de Naïve Bayes.

Estos son los resultados que obtenemos utilizando el algoritmo de Bayes. Como vemos, los resultados del test reflejan que el algoritmo ha clasificado 1 frase como positiva pero es negativa. Las frases positivas, sin embargo, las ha clasificado correctamente.

```
> recall_accuracy(tuits[11:15, 2], predicted)
[1] 0.8
```

Figura 4.8: Precisión obtenida con el algoritmo de Naïve Bayes.

En el conjunto de test que estamos realizando utilizamos el término ***recall*** como medida de la precisión. El concepto hace referencia al porcentaje de elementos que ha clasificado correctamente respecto al total de elementos que debía clasificar.

Como vemos, pese a ser uno de los algoritmos más sencillos, obtenemos unos resultados bastante buenos con una precisión del 80 %.

La salida del test ha sido:

```
1 (feel happy this morning)      - Positivo - OK
2 (larry friend)                 - Positivo - OK
3 (not like that man)            - Negativo - OK
4 (house not great)              - Negativo - OK
5 (your song annoying)           - Positivo - Falso Positivo
```

El fallo que ha tenido puede deberse a que la palabra *annoying* no se encuentra en ninguno de los tuits de entrenamiento.

Pruebas con SVM - Support Vector Machine

```
> table(as.numeric(as.factor(tuits[11:15, 2])), results[, "SVM_LABEL"])

  1 2
1 2 1
2 2 0
```

Figura 4.9: Resultados del test con el algoritmo SVM - Support Vector Machine.

Estos son los resultados que obtenemos utilizando el algoritmo de SVM. Como vemos, los resultados del test reflejan que el algoritmo ha cometido el mismo error que Bayes con las frases negativas (en este caso se indican con 1). Las frases positivas (2), sin embargo, no ha podido clasificarlas correctamente.

```
> recall_accuracy(as.numeric(as.factor(tuits[11:15, 2])), results[, "SVM_LABEL"])
[1] 0.4
```

Figura 4.10: Precisión obtenida con el algoritmo SVM - Support Vector Machine.

Como se aprecia, la precisión en este caso es de tan solo el 40 % por lo que concluimos que este algoritmo no será una buena opción. Pese a ser uno de los algoritmos más utilizados para texto, con un conjunto de entrenamiento pequeño no ha podido desenvolverse de forma correcta.

La salida del test ha sido:

```
1 (feel happy this morning)          - Negativo - Falso Negativo
2 (larry friend)                     - Negativo - Falso Negativo
3 (not like that man)                - Positivo - Falso Positivo
4 (house not great)                  - Negativo - OK
5 (your song annoying)               - Negativo - OK
```

Sin embargo, como vemos a la salida del test, los tuits con la palabra *great* y *annoying*, que no aparecen en el entrenamiento, son los únicos que ha clasificado como negativos correctamente.

Pruebas con Árboles de Decisión Básicos

```
> recall_accuracy(as.numeric(as.factor(tuits[11:15, 2])), results[, "TREE_LABEL"])
[1] 0.6
```

Figura 4.11: Precisión obtenida con Árboles de Decisión.

Si utilizamos un árbol de decisión mejoramos la precisión respecto al SVM. Sin embargo, vamos a introducir algunas mejoras (bosques de decisión, optimización basada en la entropía y bagging) sobre este algoritmo para intentar conseguir una mayor precisión.

```
1 (feel happy this morning)          - Negativo - Falso Negativo
2 (larry friend)                     - Negativo - Falso Negativo
3 (not like that man)                - Negativo - OK
4 (house not great)                  - Negativo - OK
5 (your song annoying)               - Negativo - OK
```

A la vista de la salida del test, ha clasificado todos los tuits como negativos, por lo que ha clasificado correctamente los negativos y erróneamente los positivos.

Pruebas con Árboles de Decisión - Bosques de Decisión

```
> table(tuits[11:15, 2], results[, "FORESTS_LABEL"])

          1 2
negative 3 0
positive 1 1
```

Figura 4.12: Resultados del test con Bosques de Decisión.

En este caso, utilizando un bosque de decisión, conseguimos que nuestro modelo, ahora sí, sea capaz de clasificar correctamente los textos negativos. Sin embargo, para los textos positivos el modelo no tiene la precisión necesaria.

```
> recall_accuracy(as.numeric(as.factor(tuits[11:15, 2])), results[, "FORESTS_LABEL"])
[1] 0.8
```

Figura 4.13: Precisión obtenida con Bosques de Decisión.

Utilizando un bosque de decisión, formado por varios árboles, efectivamente conseguimos mejorar la precisión.

1 (feel happy this morning)	– Positivo – OK
2 (larry friend)	– Negativo – Falso Negativo
3 (not like that man)	– Negativo – OK
4 (house not great)	– Negativo – OK
5 (your song annoying)	– Negativo – OK

Sin embargo, ha clasificado incorrectamente el segundo tuit de test que contiene la palabra *friend*, que aparece en el entrenamiento como positiva, y la palabra *larry*, que no aparece en el entrenamiento.

Pruebas con Árboles de Decisión - Optimizando con la máxima entropía

```
> table(as.numeric(as.factor(tuits[11:15, 2])), results[, "MAXENTROPY_LABEL"])

   1 2
1 2 1
2 0 2
```

Figura 4.14: Resultados del test con la optimización de máxima entropía.

En este caso, utilizamos la entropía para mejorar el modelo resultante. Realizamos las particiones que nos aporten una mayor ganancia de información.

```
> recall_accuracy(as.numeric(as.factor(tuits[11:15, 2])), results[, "MAXENTROPY_LABEL"])
[1] 0.8
```

Figura 4.15: Precisión obtenida con la optimización de máxima entropía.

La precisión es similar a la que obteníamos con un Bosque de decisión. Sin embargo, en este caso tenemos problemas con los textos negativos, aunque los positivos si que se clasifican correctamente.

1 (feel happy this morning)	– Positivo – OK
2 (larry friend)	– Positivo – OK
3 (not like that man)	– Negativo – OK
4 (house not great)	– Positivo – Falso Positivo
5 (your song annoying)	– Negativo – OK

Como vemos en la salida del test, en este caso, se ha clasificado correctamente el tuit que no conseguimos con un bosque de decisión. Sin embargo, el tuit 4 del test que contiene las palabras *house* y *great*, etiquetadas como positivas en el entrenamiento y la palabra *not*, etiquetada como negativa, se ha etiquetado erróneamente como positivo.

Pruebas con Árboles de Decisión - Optimizando con Bagging (Bootstrap Aggregating)

```
> table(as.numeric(as.factor(tuits[11:15, 2])), results[, "BAGGING_LABEL"])

   1 2
1 2 1
2 2 0
```

Figura 4.16: Resultados del test con la optimización de Bootstrap Aggregating.

En este caso se entrenan diferentes árboles que forman el bosque final y cada árbol utiliza un conjunto de datos diferentes. El resultado final se selecciona utilizando el voto mayoritario.

```
> recall_accuracy(as.numeric(as.factor(tuits[11:15, 2])), results[, "BAGGING_LABEL"])
[1] 0.4
```

Figura 4.17: Precisión obtenida con la optimización de Bootstrap Aggregating.

La precisión, en este caso, desciende considerablemente. Utilizando el Bagging se intenta reducir la varianza y evitar el overfitting del modelo. Sin embargo, en este caso, se han producido errores de clasificación para los tuits positivos y los negativos se han clasificado todos erróneamente.

```
1 (feel happy this morning)   - Negativo - Falso Negativo
2 (larry friend)              - Negativo - Falso Negativo
3 (not like that man)         - Negativo - OK
4 (house not great)           - Positivo - Falso Positivo
5 (your song annoying)        - Negativo - OK
```

La salida de este test es bastante curiosa ya que produce el mismo error que el algoritmo anterior, clasificando como positivo el tuit 4, aunque clasifica el resto como negativos; clasificando, por tanto, erróneamente los tuits positivos.

4.3.2. Conclusiones sobre la comparación de los algoritmos

```
> summary(analytics)
ENSEMBLE SUMMARY

       n-ENSEMBLE COVERAGE n-ENSEMBLE RECALL
n >= 1                 1.0              0.60
n >= 2                 1.0              0.60
n >= 3                 1.0              0.60
n >= 4                 0.6              0.67
n >= 5                 0.2              1.00


ALGORITHM PERFORMANCE

      SVM_PRECISION          SVM_RECALL          SVM_FSCORE   BAGGING_PRECISION      BAGGING_RECALL
              0.250               0.335               0.285               0.250               0.335
    BAGGING_FSCORE   FORESTS_PRECISION      FORESTS_RECALL      FORESTS_FSCORE      TREE_PRECISION
              0.285               0.875               0.750               0.765               0.300
       TREE_RECALL         TREE_FSCORE MAXENTROPY_PRECISION   MAXENTROPY_RECALL   MAXENTROPY_FSCORE
              0.500               0.375               0.835               0.835               0.800
```

Figura 4.18: Resultados generales de los tests.

Si nos fijamos en los resultados generales de los tests realizados, los ganadores son los Bosques de Decisión y la optimización con la Mayor Ganancia de Entropía.

Vamos a realizar una *4-folds cross-validation* para contrastar estos resultados realizando varias pruebas sobre distintos conjuntos de entrenamiento y tests.

```
> cross_validate(container, N, "SVM")
Fold 1 Out of Sample Accuracy = 0.6666667
Fold 2 Out of Sample Accuracy = 0.6
Fold 3 Out of Sample Accuracy = 1
Fold 4 Out of Sample Accuracy = 0
[[1]]
[1] 0.6666667 0.6000000 1.0000000 0.0000000

$meanAccuracy
[1] 0.5666667
```

(4-folds cross-validation para SVM)

```
> cross_validate(container, N, "TREE")
Fold 1 Out of Sample Accuracy = 0.3333333
Fold 2 Out of Sample Accuracy = 0.3333333
Fold 3 Out of Sample Accuracy = 0.25
Fold 4 Out of Sample Accuracy = 0.6
[[1]]
[1] 0.3333333 0.3333333 0.2500000 0.6000000

$meanAccuracy
[1] 0.3791667
```

(4-folds cross-validation para Árboles)

```
> cross_validate(container, N, "RF")
Fold 1 Out of Sample Accuracy = 0.6
Fold 2 Out of Sample Accuracy = 0.3333333
Fold 3 Out of Sample Accuracy = 0.4
Fold 4 Out of Sample Accuracy = 1
[[1]]
[1] 0.6000000 0.3333333 0.4000000 1.0000000

$meanAccuracy
[1] 0.5833333
```

(4-folds cross-validation para Random Forest)

```
> cross_validate(container, N, "MAXENT")
Fold 1 Out of Sample Accuracy = 1
Fold 2 Out of Sample Accuracy = 1
Fold 3 Out of Sample Accuracy = 0.8571429
Fold 4 Out of Sample Accuracy = 1
[[1]]
[1] 1.0000000 1.0000000 0.8571429 1.0000000

$meanAccuracy
[1] 0.9642857
```

(4-folds cross-validation para Máxima Entropía)

Cuadro 4.1: 4-folds cross-validation sobre los diferentes algoritmos

Claramente, destaca la optimización con Máxima Entropía de un Árbol de decisión arrojando una media de exactitud del 96.43 % en los tests.

Vamos a intentar realizar una última prueba con un conjunto de tuits mayor. Utilizaremos un conjunto llamado *Twitter US Airline Sentiment* [12] que contiene 14500 tuits sobre las aerolíneas clasificados como positivos, negativos o neutrales. En este consideramos 1 negativo, 2 neutral y 3 positivo.

```
Large matrix
(263080800
elements, 2 Gb)
```

(Tamaño de la matriz de términos-documentos)

```
Error: protect(): protección del desborde de la pila
```

(Error de desbordamiento de la pila)

El primer problema que nos aparece es al generar la matriz de términos - documentos. El tamaño se dispara rápidamente llegando a alcanzar los 263080800 términos (columnas) y ocupando 2 Gb en memoria.

La gran cantidad de información a procesar provocará en algunos algoritmos un *stack overflow* (desbordamiento de la pila) debido a la gran cantidad de llamadas recursivas que se producen. Además, se necesita mucho tiempo para procesar todos los datos, por lo que optaremos por reducir el conjunto de datos.

Utilizamos 1000 tuits para entrenamiento y 400 tuits para los tests. Conseguimos entrenar un SVM, Naïve Bayes y la optimización con Máxima Entropía en un tiempo razonable y sin el desbordamiento de pila y éstos son los resultados:

```
> table(Tweets$airline_sentiment[1000:1400], predicted)
          predicted
           negative neutral positive
  negative        0       0      290
  neutral         0       0       62
  positive        0       0       49
```

(Resultados del test con Naïve Bayes)

```
> recall_accuracy(Tweets$airline_sentiment[1000:1400], predicted)
[1] 0.1221945
```

(Precisión obtenida con Naïve Bayes)

Cuadro 4.2: Tests con Naïve Bayes

Utilizando Naïve Bayes, en este caso (4.2), obtenemos una baja precisión del 12,22 %. El algoritmo ha clasificado todos los tuits como positivos y no se ha comportado tan bien como con el conjunto de tuits pequeño que hemos utilizado en las pruebas anteriores.

```
> table(as.numeric(as.factor(Tweets$airline_sentiment[1000:1400])), results[, "SVM_LABEL"])

     1   2   3
1 236  38  16
2  19  32  11
3   9  11  29
```

(Resultados del test con SVM)

```
> recall_accuracy(as.numeric(as.factor(Tweets$airline_sentiment[1000:1400])), results[, "SVM_LABEL"])
[1] 0.7406484
```

(Precisión obtenida con SVM)

Cuadro 4.3: Tests con SVM

Con SVM, en este caso (4.3) , mejoramos la precisión respecto a las pruebas anteriores con un conjunto de datos más reducido. Obtenemos una precisión del 74,06 % y, si nos fijamos en los resultados del test, vemos que el algoritmo clasifica correctamente la mayor parte de los tuits de cada tipo (positivo, negativo y neutral).

```
> table(as.numeric(as.factor(Tweets$airline_sentiment[1000:1400])), results[, "MAXENTROPY_LABEL"])

     1   2   3
  1 222  40  28
  2  21  23  18
  3  18   7  24
```

(Resultados del test con Máxima Entropía)

```
> recall_accuracy(as.numeric(as.factor(Tweets$airline_sentiment[1000:1400])), results[, "MAXENTROPY_LABEL"])
[1] 0.6708229
```

(Precisión obtenida con Máxima Entropía)

Cuadro 4.4: Tests con Máxima Entropía

Finalmente, en las pruebas con el algoritmo de Máxima Entropía (4.4), obtenemos una precisión del 67,08 %; menor que la precisión que obtuvimos con el test de tamaño reducido y menor que la precisión obtenida con SVM.

Por lo tanto, ahora sí, podemos afirmar que el algoritmo de SVM será el que mejores prestaciones nos ofrece cuando busquemos realizar una clasificación de texto.

4.3.3. Análisis de sentimientos de tuits en español

Como hemos visto en la sección anterior, entrenar y probar un modelo es una tarea larga y tediosa. Necesitamos generar un conjunto de datos de entrenamiento compuesto por tuits y su correspondiente etiqueta de sentimiento (positivo, negativo o neutral). Por ello, para realizar el análisis de los sentimientos de los tuits en español, utilizamos la librería *syuzhet*, que incluye unos lexicons (listados de palabras) en español para la clasificacción de sentimientos en positivos, negativos y neutrales.

En este caso utilizaremos el texto de las publicaciones (tuits) preprocesado para eliminar menciones y hashtags ya que dichas palabras no nos aportan información para realizar un análisis de los sentimientos y podrían influir en los resultados.

Primero obtenemos un vector (listado) con las palabras que nos aportan información (después del preprocesado) de cada uno de los tuits y lo enviamos al modelo de *syuzhet*. El modelo ha sido preentrenado con frases extraídas de la literatura española y nos permitirá puntuar cada uno de los vectores (tuits) con valores positivos o negativos según el sentimiento.

```
emotions.df <- get_nrc_sentiment(palabras.df)
emotions.df2 <- cbind(tuits.df_limpio, emotions.df)
sent.value <- get_sentiment(palabras.df)
```

A continuación filtramos los tuits en los más positivos y negativos y los clasificamos. Un valor de sentimiento < 0 nos indica un sentimiento negativo y un valor > 0 un sentimiento positivo. Los tuits neutrales tienen un valor $= 0$.

```
most.positive <- palabras.df[sent.value == max(sent.value)]
most.negative <- palabras.df[sent.value == min(sent.value)]
positive.tweets <- palabras.df[sent.value > 0]
negative.tweets <- palabras.df[sent.value < 0]
```

```
> most.negative
[1] "@balbuenajm: El sistema de #PrimariasPP debe de ser reformado. A ver si lo entiendo. De Cospedal salió derrotada, sin emba
rgo, puede con…"
```

Figura 4.19: Tuit más negativo de *#primariasPP*.

Finalmente, generamos una tabla a modo resumen con la cantidad de tuits positivos y negativos.

```
category_senti <- ifelse(sent.value < 0, "Negative",
                  ifelse(sent.value > 0, "Positive", "Neutral"))
```

```
> table(category_senti)
category_senti
Negative  Neutral Positive
    1062     6379      740
>
```

Figura 4.20: Análisis de sentimientos de las publicaciones de *#primariasPP*.

4.3.4. Conclusiones sobre la aplicación

Por una parte, el análisis de los términos (sustantivos y adjetivos) que aparecen con más frecuencia nos revela que palabras como *votos*, *victoria*, *República* o *renovación* y *ppeppera*, *imputado*, *amañado* o *joven* son las más utilizadas. Además, cuales son los principales temas sobre los que han hablado los usuarios en relación con el hastag #PrimariasPP.

Por otra parte, el análisis de coocurencia (aparición de palabras emparejadas) de los términos extraídos de las publicaciones y la aplicación del algoritmo RAKE nos muestra cuales son las

palabras clave que más se han utilizado y que términos han aparecido emparejados. Palabras como *gran ambiente*, *gran perdedora*, *clara renovación*, *urbanismo imputado*, *dinero exconcejal* o *imputado viaje* nos permiten conocer o sintetizar cuáles son los principales temas sobre los que la población interesada en #PrimariasPP ha opinado.

Finalmente, hemos probado los diferentes algoritmos para el análisis de sentimientos. Lo hemos aplicado a un conjunto de publicaciones de Twitter y esto nos permite clasificar en positivas y negativas las publicaciones y conocer así cuáles han sido los peores y mejores comentarios sobre el evento. Con una lectura de los resultados, los responsables del evento pueden detectar cuáles han sido los principales problemas o quejas que la población ha tenido y cuáles han sido los puntos fuertes que han sido agradecidos. En general se puede ver una mayor cantidad de publicaciones negativas en relación a #PrimariasPP por lo que podemos concluir que la población no ha acogido con agrado dicho evento y han habido muchas quejas al respecto, 740 positivos frente a 1062 negativos.

Capítulo 5

Conclusiones

Comenzamos introduciendo el concepto de minería de datos como un aprendizaje computacional inductivo; que nos permite predecir y describir ciertos datos de entrada de forma automática. Explicamos los diferentes tipos de problemas de aprendizaje que podían presentarse; los de Interpolación, los Secuenciales y los de Aprendizaje Supervisado o No Supervisado.

Aprendimos a analizar asociaciones entre los conjuntos de datos y descubrir patrones frecuentes. Esto nos permitió buscar itemsets frecuentes y reglas de asociación fuertes. También hablamos sobre los algoritmos de clustering para agrupar; bien sea particionando, basándonos en la densidad o generando dendogramas de agrupación jerárquicos de forma aglomerativa y divisiva.

A continuación, aprendimos los algoritmos de clasificación y prediccion; desde Naïve Bayes y SVM hasta las Estructuras en Forma de Árbol con sus algoritmos y optimizaciones.

Finalmente, entrenamos y probamos cada uno de los modelos y los utilizamos para procesar texto procedente de los tuits, que extraemos de un hashtag, para obtener información sobre un evento o una entidad.

Además, durante la estancia en prácticas, implementamos una app web que también procesa las publicaciones de redes sociales para detectar clientes potenciales utilizando técnicas de minería de datos.

Bibliografía

[1] Hans-Peter; Sander Jörg; Xu Xiaowei Ester, Martin; Kriegel. A density-based algorithm for discovering clusters in large spatial databases with noise. [Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining] (1996).

[2] Elizabeth León Guzman. Teoría estadística bayesiana. [Universidad Nacional de Colombia].

[3] M. Han, J; Kamber. Data mining: Concepts and techniques. [The Morgan Kaufmann Series in Data Management Systems].

[4] https://www.r project.org. The r project for statistical computing. [Software gratuito para computación estadística].

[5] L. Kaufman and P.J. Rousseeuw. Clustering by means of medoids, in statistical data analysis based on the l_1–norm and related methods. [edited by Y. Dodge, North-Holland] (1987).

[6] Nebraska Literary Lab. syuzhet - extracts sentiment and sentiment-derived plot arcs from text.

[7] David Meyer. Paquete rtexttools con bayes. [Department of Statistics (e1071)] TU Wien.

[8] Sargantano. [cc by-sa 3.0 (https://creativecommons.org/licenses/by-sa/3.0)].

[9] Milan Straka; Jana Straková. Udpipe - a trainable pipeline for tokenization, tagging, lemmatization and dependency parsing of conll-u files. [Institute of Formal and Applied Linguistics - Charles University, Czech Republic - Faculty of Mathematics and Physics].

[10] Enrique J. Carmona Suárez. Tutorial sobre máquinas de vectores soporte (svm). [Universidad Nacional de Educación a Distancia (UNED)].

[11] Jerome Friedman Trevor Hastie, Robert Tibshirani. The elements of statistical learning: Data mining, inference, and prediction. [Second Edition (Springer Series in Statistics)].

[12] www.kaggle.com. The world's largest community of data scientists and machine learners.

[13] R. Agrawal y R.Srikant. Algoritmo apriori. [1994].